XLBlocks: On the Effect of a Visual Language on Formula Creation and Comprehension in Spreadsheets

XLBlocks: On the Effect of a Visual Language on Formula Creation and Comprehension in Spreadsheets

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof. dr. ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen
op maandag 12 september 2022 om 17:30 uur

door

Bas JANSEN

Master of Science in Industrial Engineering & Management,
Twente University Enschede, The Netherlands,
geboren te Arnhem, Nederland.

Keywords: End-User Programming, Spreadsheets, Block-based languages

Printed by:

Cover:

Style: TU Delft House Style, with modifications by Moritz Beller
`https://github.com/Inventitech/phd-thesis-template`

The author set this thesis in LaTeX using the Libertinus and Inconsolata fonts.

ISBN ...

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/`.

There is no better motivation to finish a dissertation than when a loving father, a wonderful wife, and an involved supervisor conspire against you.

Contents

Summary

The use of spreadsheets in industry is widespread. Their outcomes are often used to substantiate critical business decisions. Unfortunately, spreadsheets are also error-prone. Consequently, companies risk making decisions based on inaccurate information, which eventually could lead to loss of money and reputation.

This dissertation investigates to what extent a visual language could support professional spreadsheet users in interacting with complex formulas. We divided our research into two phases. In the first phase, we try to understand better how spreadsheets are used in three ways. We looked at:

1. code smells in formulas
2. the influence on comprehension of how data is structured in a spreadsheet
3. how spreadsheets evolve.

In the second phase, we developed XLBlocks: a block-based formula editor for spreadsheets.

Phase I

Code smells

Based on a data set from financial modeling company F1F9, we researched the occurrence of formula smells in spreadsheets. The data set consisted of 54 pairs of spreadsheets. Each pair consisted of a model supplied by a customer of F1F9 and an improved model rebuilt from scratch by professional modelers of F1F9. We hypothesized that the improved models should contain fewer smells, and that was indeed the case. However, we also found that size and coupling metrics are not good indicators to predict the complexity of a spreadsheet. The professional F1F9 models scored low on the occurrence of smells, but some still scored high on the size and coupling metrics.

Structure of data

We applied the concept of delocalized plans to spreadsheets to study how data is structured in a spreadsheet affects the comprehensibility of a spreadsheet. We defined a delocalized plan in a spreadsheet as a formula that has its precedents spread widely across the spreadsheet.

We conducted a controlled experiment with 107 spreadsheet users. We asked them to execute a set of comprehension tasks on a spreadsheet model. We used two different variants of the same spreadsheet. One group received a spreadsheet where the data necessary for a calculation was located close to the calculation itself. The other group got the same calculations, but now the data used in the calculation was grouped by category and, as a result, spread far and wide across the spreadsheet.

We found that the participants performed better in explaining formulas using the spreadsheet model where the data was closely located to the calculation. We also observed that when participants have to make changes to a formula, they perform better on a longer formula with a relatively short calculation chain than on a short formula with (as a consequence) a longer calculation chain.

Spreadsheet evolution

Spreadsheets have an average lifespan of five years. During their existence, they are maintained and, consequently, evolve. To better understand how spreadsheets evolve, we conducted two case studies on two different sets of spreadsheets that both were already maintained for three years. To study the evolution, we developed FormulaMatch, an algorithm to detect and visualize changes in spreadsheets.

From the case studies, we learned that spreadsheets grow over time both in data and in number of formulas. New feature requests mainly drive the growth. Also, during the lifespan of a spreadsheet, formulas will be changed. In the case studies, we saw that from version to version, usually, the same formulas are changed. These changes are often necessary because new data is added to the spreadsheet. Other reasons for change are improving the maintainability of the spreadsheet and bug fixing.

Phase II

XLBlocks

During the second phase of our research, we developed XLBlocks: a block-based formula editor for spreadsheet formulas. In the first version of the language, it was possible to generate valid Excel formulas from a block-based model of the formula created in the visual editor of XLBlocks.

We conducted a think-aloud study, asking participants to perform a set of typical spreadsheet tasks with XLBlocks. Then we interviewed them and asked them to evaluate XLBlocks, using the Cognitive Dimensions of Notation (CDN) framework.

XLBlocks received a better score than the Excel formula editor on all dimensions. Users appreciated XLBlocks because they did not have to worry about the syntax of a formula, the drag and drop interface made it easier to edit formulas, and they had the freedom to develop the formula in the order they saw fit.

After the first think-aloud study, we further developed XLBlocks and extended it with the functionality to generate a block-based representation of an existing (textual) formula. We organized a second think-aloud study that focused on spreadsheet comprehension.

Results from the study indicate that participants believed that XLBlocks helped them comprehend the spreadsheet better. They gave several reasons for this. The visualization in blocks supported them in splitting the formulas into smaller parts in their head, making them easier to comprehend. Furthermore, the labels used in the blocks make it easier to read a formula, and finally, the highlighting functionality of XLBocks, combined with the functionality to navigate the formula precedents, makes it easier to understand the spreadsheet as a whole.

Samenvatting

Spreadsheets worden volop gebruikt in het bedrijfsleven. De uitkomsten hiervan worden vaak gebruikt bij belangrijke zakelijke beslissingen. Helaas zijn spreadsheets foutgevoelig. Hierdoor lopen bedrijven het risico dat ze hun beslissingen baseren op onjuiste informatie. Uiteindelijk kan dat leiden tot financiële verliezen en/of reputatieschade.

Dit proefschrift onderzoekt in hoeverre een visuele taal professionele spreadsheetgebruikers kan ondersteunen bij het maken en begrijpen van complexe formules. We hebben ons onderzoek onderverdeeld in twee fasen. In fase I hebben we geprobeerd om vanuit drie invalshoeken beter te begrijpen hoe spreadsheets worden gebruikt. We hebben hierbij gekeken naar:

1. het voorkomen van zogenaamde 'code smells' in spreadsheetformules
2. de vraag of de manier waarop de gegevens in een spreadsheet zijn georganiseerd van invloed is op de begrijpbaarheid van een spreadsheet
3. de evolutie van spreadsheets

In fase II van het onderzoek hebben we XLBlocks ontwikkeld. XLBlocks is een visuele, op blokken gebaseerde, programmeertaal voor het maken en onderhouden van formules in spreadsheets.

Fase I

Code smells

Met behulp van een dataset met spreadsheets van de financiële modellenbouwer F1F9 hebben we gekeken hoe vaak code smells voorkomen in spreadsheetformules. De dataset bestond uit 54 spreadsheetparen. Ieder paar bestond uit een spreadsheet aangeleverd door een klant van F1F9 en een door de professionele modelbouwers van F1F9 verbeterde variant van diezelfde spreadsheet. We vermoedden dat de door de professionele modelbouwers ontwikkelde spreadsheets minder code smells zouden bevatten en dit bleek inderdaad het geval te zijn. Daarnaast bleek dat kengetallen voor omvang en koppelingsdichtheid (het aantal onderlinge verbindingen tussen formules) slechte voorspellers zijn van de complexiteit van een spreadsheet. Er waren geen duidelijke verschillen voor deze kengetallen tussen de spreadsheets van de klant en die van F1F9.

Organisatie van de gegevens in een spreadsheet

We hebben het concept van 'delocalized plans' toegepast op spreadsheets. Delocalized plans in spreadsheets hebben we gedefinieerd als formules waarbij de onderdelen van de berekening van die formule ver van elkaar verwijderd staan in de spreadsheet. We hebben onderzocht in hoeverre de aanwezigheid van delocalized plans van invloed is op de begrijpbaarheid van de spreadsheet.

Hiertoe hebben we een gecontroleerd experiment opgezet. We hebben de 107 deelnemers gevraagd om een aantal taken uit te voeren in een spreadsheet. Deze taken konden alleen goed uitgevoerd worden als degene die de taak uitvoerde de spreadsheet goed begreep. In het experiment hebben we twee varianten van dezelfde spreadsheet gebruikt. De ene groep deelnemers moest de taak uitvoeren in een spreadsheet waarbij de gegevens voor een berekening zo dicht mogelijk bij elkaar stonden. De andere groep moest dezelfde taken uitvoeren in een andere variant van de spreadsheet waarbij de gegevens voor de berekening verspreid stonden over de spreadsheet.

Het bleek dat deelnemers formules beter konden uitleggen als de gegevens die gebruikt werden in de formule dicht bij elkaar stonden. Ook werd duidelijk dat ze beter in staat waren om veranderingen in formules aan te brengen als die formule langer was en daardoor een kortere ‘calculation chain’ had. Bij kortere formules met een langere calculation chain werden meer fouten gemaakt.

Spreadsheet evolutie

De gemiddelde levensduur van een spreadsheet is vijf jaar. Gedurende deze periode worden ze regelmatig onderhouden en aangepast. Om meer te weten te komen over hoe spreadsheets evolueren, hebben we twee casestudies uitgevoerd op twee verschillende sets van spreadsheets. Beide sets bestonden al 3 jaar. Om de evolutie te kunnen onderzoeken was het nodig om de verschillen tussen twee versies van een spreadsheet vast te kunnen stellen. Hiervoor hebben we FormulaMatch ontwikkeld. Het is een algoritme dat verschillen tussen versies opspoort en visualiseert.

Op basis van de casestudies werd duidelijk dat spreadsheets gedurende hun leven blijven groeien. Het gaat hierbij om groei van zowel gegevens als het aantal formules. Belangrijkste reden voor deze groei zijn nieuwe stukjes functionaliteit die worden toegevoegd aan de spreadsheet. Ook bleek dat er regelmatig formules worden aangepast. Van versie op versie zijn het veelal dezelfde formules die worden aangepast. De reden voor deze aanpassingen is vaak dat er nieuwe gegevens zijn toegevoegd aan de spreadsheet. Andere redenen voor aanpassingen in formules zijn het verbeteren van de onderhoudbaarheid en het corrigeren van fouten.

Fase II

XLBlocks

In de tweede fase van ons onderzoek hebben we ons gericht op het ontwikkelen van XLBlocks: een op blokken gebaseerde programmeertaal voor het maken en aanpassen van spreadsheetformules. In de eerste versie van de taal was het mogelijk om een blokmodel te maken van een formule en deze om te laten zetten naar een geldige Excelformule.

Om meer te leren over het gebruik van de taal hebben we een ‘think-aloud study’ opgezet. We gaven de deelnemers een aantal typische spreadsheetopdrachten die ze moesten oplossen met XLBlocks. Daarna hebben we ze een interview afgenomen waarin we ze gevraagd hebben naar hun ervaringen met XLBlocks. Hierbij hebben we gebruik gemaakt van het Cognitive Dimensions of Notation (CDN) framework.

XLBlocks kreeg van de deelnemers voor alle CDN-dimensies een hogere score dan de standaard formulebalk in Excel. Ze vonden het prettig dat ze met XLBlocks niet hoefden na te denken over de precieze syntax van een formule, de ‘drag-and-drop’ interface maakte het eenvoudiger om formules aan te passen en XLBlocks gaf deelnemers de vrijheid om de formule op te bouwen in de volgorde die zij logisch vonden.

Na de eerste studie hebben we XLBlocks verder doorontwikkelt. We hebben de mogelijkheid toegevoegd om een bestaande Excelformule om te zetten naar een blokmodel van die formule in XLBlocks.

Ook voor deze tweede versie van XLBlocks hebben we een think-aloud studie uitgevoerd. In deze studie hebben we met name gekeken naar het effect van een visuele taal op de begrijpbaarheid van een spreadsheet.

Uit de uitkomsten van de studie komt naar voren dat deelnemers het gevoel hebben dat XLBlocks het makkelijker maakt om een spreadsheet te doorgronden. Ze geven hiervoor een aantal redenen. Allereerst helpt de visualisatie van een formule om deze mentaal op te delen in kleinere stappen waardoor deze makkelijker te begrijpen is. Daarnaast maken de omschrijvingen van de parameters van de functie het makkelijker om de formule te lezen. Tot slot maakt de combinatie van het markeren van formules in de spreadsheet en de mogelijkheid om eenvoudig van formule naar formule te navigeren, het eenvoudiger om de spreadsheet in z’n geheel te begrijpen.

Acknowledgments

Finishing a Ph.D. is not a sprint but a marathon and, in my case, a long one. Nevertheless, I enjoyed every bit of the race. And I could only enjoy it because of all the people that surrounded and supported me. In the following paragraphs, I will seriously try to thank you all.

Hermien: I will start with you. You are my soulmate, my Northstar, and you supported me from start to finish. It was not easy. Simultaneously pursuing a Ph.D. and running one's own company takes a severe toll on one's private life. You were not happy with that, but still, you realized it was important to me and supported me unconditionally. You had my back every time I dropped the ball. Without you, I would have never made the finish. I can not thank you enough for this. I owe you.

Ella: When I started my Ph.D., you were a little girl of seven, now a young woman of 15 and my paranymph. You can not imagine how proud I am of you. I thank you for being a constant source of joy in my life, and I apologize for all the times dad was not available because he had to write a paper.

Dad: You were undoubtedly my biggest supporter. You were there at the start and would have almost given anything to witness the finish. Unfortunately, it was not meant to be. We had to let you go. I promised you that I would finish this project, and today I can fulfill this promise. Thank you for giving me a solid foundation in life. You taught me to always go the extra mile. Thank you.

Mom: You accompanied me for the first 15 years of my life, and you could not have imagined that I would start a Ph.D. one day. I'm convinced you gave me the curiosity and eagerness to learn that drove me into this project.

Felienne: You are the best co-promoter one can wish for. Boy, did I learn a lot from you. You always found the right balance between pushing me and giving me slack. I remember a specific paper where I was determined to tell you I would not submit it. I just had to inform you by phone. The call lasted about 15 minutes, and somehow, by the end, you convinced me to finish the paper. I still do not understand how. However, I did finish the paper and submitted it. It was accepted and now is chapter five of this thesis.

The most important thing I learned from you is to tell my story. Answer the why and convince the reader already in the introduction of the story. Valuable lessons, not only for academic writing but for life in general.

I certainly will miss our running routine. Every Friday, we would start early in the morning with a run for 45 - 60 minutes. We would discuss, among many other things, the progress of my research. To keep me talking was a smart strategy to keep up with my running pace in the early days Nowadays, that's not necessary anymore. You easily outrun me.

Arie: Thank you for being my promotor and allowing me to pursue my Ph.D. in your group. You had a bit of a laissez-faire strategy with me. But there were a few instances I really needed your advice, and then you were there for me. Your advice was always wrapped in a question. A question that would make me think, and eventually, I would stumble on the answer. Thank you for your guidance.

Edwin: You are a definite constant in my academic life. Long ago, you guided me in writing my master thesis at Kema, you allowed me to do research for my Ph.D. at Alliander (and because of that, co-authored chapter 4 with me), helped me to sharpen my propositions, and at the finish line of this journey, you support me by being my paranymph. Your advice is sincere, well thought out, fact-based, and nuanced. Thank you for your support, but above all, for being my friend.

Fenia, Alaaeddin, Sohon, and Moritz: The first years of my Ph.D. took place in Delft, and you were an integral part of this time. I enjoyed working and spending time with you. I have fond memories of the international diner at Fenia's place and barbecue at Mortiz's place. Going out for lunch with you was one of the highlights of my Fridays. Not for the food (although I spotted a pattern there), but for the many discussions we had about a variety of topics. We all come from different cultural backgrounds, which made the conversations much more interesting. In most cases, it gave me new insights and made me a more humble world citizen. Thank you for that.

Petra: You have been my business partner since just before my Ph.D. I remember asking you if I should go for it, and you immediately said yes. And even though it took away time and energy from the company, you've always supported me. Always interested in what I was up to, and if you somehow could participate, you would. Thanks.

And last but not least, I want to thank all the people that participated in one of the different studies. The 107 spreadsheet users that participated in the experiment about delocalization in spreadsheets (Chapter 3), The employees at Alliander that participated in the case studies (Chapter 4), the 13 spreadsheet users that evaluated the first version of XLBlocks (Chapter 5), and the 21 spreadsheet users that participated in the second study about XLBlocks (Chapter 6). There are too many of you to mention you all by name, but research is impossible without you. Your valuable feedback is what brings progress. You gave me the most precious thing you possess, your time. Thank you for that.

Nine years is a long time, if you in someway helped or supported me in this endeavor and I did not mention you, let me buy you a beer the next time I see you.

Bas
Banff National Park, Canada, July 2022
metronome.unplanned.curfew

1

Introduction

1.1 Background

Spreadsheet usage
Spreadsheets are widely used in industry. According to Panko and Ordway [1] 95% of U.S. firms use spreadsheets in some form of financial reporting. Panko and Ordway also interviewed 118 business leaders, and 85% of them stated that they use spreadsheets in financial reporting and forecasting. Winston [2] found that 90% of all analysts use spreadsheets for their calculations. Furthermore, a study by the USA Bureau of Labor Statistics in 2003 [3] showed that 60% of 77 million surveyed workers in the U.S. reported that using spreadsheets is the third common use of computers after e-mail and word processing. In another study among 95 organizations in Europe, North America, Australia, and Asia, spreadsheets were placed fourth, after e-mail, browsing, and word processing [4].

Not only are spreadsheets widely used, but they are also used to support critical business decisions. Croll interviewed 23 professionals that were working in the City of London, who stated that spreadsheets are used to value financial instruments of all types and guide decisions on what to trade and when. Caulkins *et. al.* [5] came to a similar conclusion. A study among 45 executives and senior managers affirmed that spreadsheets are frequently used to inform decisions. Furthermore, Hermans *et. al.* [6] interviewed 27 analysts of an asset management company and more than half of them stated that they use calculations in spreadsheets as a basis for their decisions.

Spreadsheet errors
From research, we also know that spreadsheets are error-prone. The cell error rate is a metric to measure the error-proneness of a spreadsheet. It is the number of errors divided by the combined number of numerical and formula cells [7]. Several studies [8] [9] reveal an average cell error rate between one and five percent. In the Enron corpus (15,770 spreadsheets), Hermans and Murphy-Hill [10] found an average of 1,286 formula cells per spreadsheet. Combined with a cell error rate of one percent means, on average, thirteen erroneous cells in a spreadsheet. Based on these numbers, it is expected that almost every spreadsheet in the Enron corpus contains errors, which agrees with a finding from Panko that 86% of the spreadsheets contain errors [11].

These error rates are not specific for spreadsheets but consistent with human error rates from other work domains. Depending on the complexity of the tasks, humans tend to make undetected errors in about 0.5% to 5% of their actions [7]. The difference is the impact of these errors. If a user makes a typo while writing a report, the typo does not change the report's outcome. However, if a user makes a typo in a formula, it will most likely change the spreadsheet's calculation outcome.

The widespread use of spreadsheets to support critical business decisions combined with their error proneness leads to the risk that companies make decisions based on inaccurate information, leading to loss of money and reputation.

An infamous example of reputation loss caused by a spreadsheet error is the Reinhart and Rogoff controversy. Based on their study Reinhart and Rogoff concluded that once the public-debt-to-GDP ratio rises above 90%, the average economic growth rate is a negative 0.1% [12]. The conclusions of this study were frequently quoted on the issue of whether a Keynesian stimulus program or policy of austerity would be the appropriate reaction to the economic crisis of 2008.

In a replication study, Herndon *et. al.* found several errors in Reinhart and Rogoff's study, one of them being a spreadsheet coding error that led to the exclusion of growth data of five countries [13]. When properly calculated, Herndon *et. al.* found that the average GDP growth rate for countries carrying a public-debt-to-GDP ratio over 90 percent is 2.2% instead of Reinhart and Rogoff's -0.1%, meaning economic growth instead of decline.

The delay of the opening of Edinburgh's new children's hospital illustrates the possible financial impact of a mistake in a spreadsheet [1]. Critical care rooms need ten air changes per hour, but due to the spreadsheet error, they were outfitted with a ventilation system that only did four air changes per hour. This led to remedial work worth sixteen million GBP that had to be carried out to correct the ventilation system.

An extensive list of spreadsheet horror stories illustrating the above mentioned risks can be found on the website of the European Spreadsheet Risk Interest Group[2].

Spreadsheets are code

Hermans *et. al.* [14] state that spreadsheets can be considered to be the world's most successful end-user programming language and in their paper they give several compelling reasons why spreadsheets are code:

- They share similar goals. Spreadsheets are used to solve similar problems like complex financial calculations or data manipulation.

- They have comparable expressive power and share concepts like composition, selection, and repetition. Examples of composition are formulas that reference other cells in the spreadsheet. Selection is implemented by providing an if-then-else function. Repetition is found in the replication of the same formula across many rows or columns.

- They share maintainability issues with software. They have a long lifespan (an average of five years), are used by many different users (an average of twelve different users), lack documentation [6], and suffer, because of the error-proneness, from quality issues.

Motivation

The similarities mentioned above between spreadsheets and software motivate to seek inspiration from methods and techniques in software engineering to improve spreadsheets. *The primary motivation for this dissertation is the wish to reduce the error-proneness of spreadsheets and, consequently, reduce the risks of loss of money and reputation by companies.* Can we use the acquired knowledge in the field of software engineering to improve spreadsheets?

[1]https://www.bbc.com/news/uk-scotland-edinburgh-east-fife-53893101
[2]http://www.eusprig.org/horror-stories.htm

1.2 Spreadsheet challenges

Although spreadsheets have similarities with code, they also possess some properties that make them different. These properties can be divided into three categories: language-related, user-related, or tool-related. We will elaborate on these properties in the coming paragraphs.

1.2.1 Language

The first property is the *directness* of spreadsheets. In spreadsheet development, there is no difference between coding and runtime. A change in a formula results immediately in a different outcome of the spreadsheet model. Furthermore, a formula is visually replaced by its result as soon as the user confirms it. This effect further enhances the sense of directness. The directness brings the user of the spreadsheet the advantage of high interactivity, but it also hides the design of the spreadsheet behind the results of the spreadsheet, which makes it more challenging to comprehend the spreadsheet model.

Almost all programming languages accommodate the concept of a loop with statements like `for...next` and `do...while`. Spreadsheets do not have such looping constructs. Spreadsheet users copy formulas to execute the same calculation for different values to reach the same effect. It is not uncommon that a single formula is copied thousands of times. These ranges of similar formulas introduce potential errors. If one formula is changed, the spreadsheet software will not automatically populate this change to the copied formulas. If the user forgets to do this manually, an error is introduced in the spreadsheet.

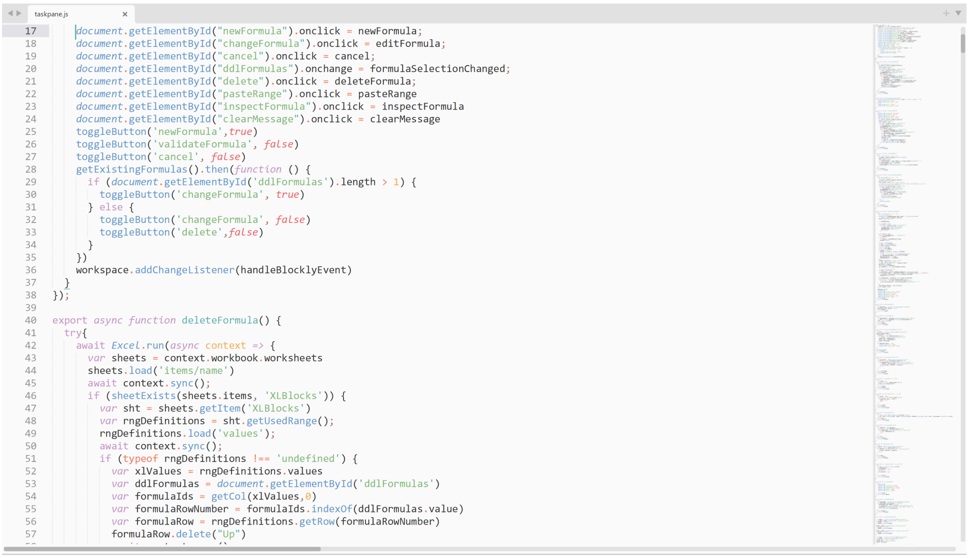
In a spreadsheet interface, it is impossible to see which formulas are a copy of each other. With a high number of copied formulas, it could be challenging to discover the unique formulas that define the business rules in the spreadsheet. This makes it more challenging to comprehend a spreadsheet model.

1.2.2 Tools

An important tool for software developers is the Integrated Development Environment (IDE). It can be seen as a sophisticated text editor specially designed to write code (See also Figure 1.1). There are many different IDEs, but in general, IDEs support the developer by providing functionality including:

- Syntax highlighting: the editor displays source code in different colors according to the category of terms.
- Refactoring: the process of restructuring source code without changing its external behavior with the intention to improve the code.
- Minimap: a reduced overview of the entire file, displayed in a separate pane, typically next to the source code.
- Debugger: from within the editor, it is possible to run the source code under controlled conditions, track its progress, and halt it at specific points.
- Autocomplete: feature in an editor that predicts the rest of a word or snippet while the user is typing.

- Version management: a system that tracks and provides control over changes in the program.



```
    document.getElementById("newFormula").onclick = newFormula;
    document.getElementById("changeFormula").onclick = editFormula;
    document.getElementById("cancel").onclick = cancel;
    document.getElementById("ddlFormulas").onchange = formulaSelectionChanged;
    document.getElementById("delete").onclick = deleteFormula;
    document.getElementById("pasteRange").onclick = pasteRange
    document.getElementById("inspectFormula").onclick = inspectFormula
    document.getElementById("clearMessage").onclick = clearMessage
    toggleButton('newFormula',true)
    toggleButton('validateFormula', false)
    toggleButton('cancel', false)
    getExistingFormulas().then(function () {
      if (document.getElementById('ddlFormulas').length > 1) {
        toggleButton('changeFormula', true)
      } else {
        toggleButton('changeFormula', false)
        toggleButton('delete',false)
      }
    })
    workspace.addChangeListener(handleBlocklyEvent)
  }
});

export async function deleteFormula() {
  try{
    await Excel.run(async context => {
      var sheets = context.workbook.worksheets
      sheets.load('items/name')
      await context.sync();
      if (sheetExists(sheets.items, 'XLBlocks')) {
        var sht = sheets.getItem('XLBlocks')
        var rngDefinitions = sht.getUsedRange();
        rngDefinitions.load('values');
        await context.sync();
        if (typeof rngDefinitions !== 'undefined') {
          var xlValues = rngDefinitions.values
          var ddlFormulas = document.getElementById('ddlFormulas')
          var formulaIds = getCol(xlValues,0)
          var formulaRowNumber = formulaIds.indexOf(ddlFormulas.value)
          var formulaRow = rngDefinitions.getRow(formulaRowNumber)
          formulaRow.delete("Up")
```

Figuur 1.1: Example of an IDE with examples of syntax highlighting and a minimap

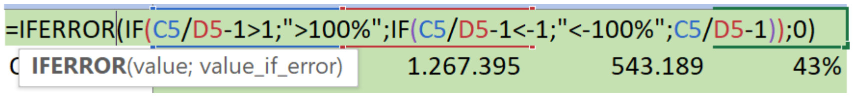
Spreadsheet software does not have an IDE for writing formulas. In the most used spreadsheet program Microsoft Excel a user can enter a formula in three different ways:

1. directly in a cell (See Fig. 1.2a)
2. in the formula bar (See Fig. 1.2b)
3. via the function wizard (See Fig. 1.2c)

In comparison with an IDE, there are seven issues with these different input methods for spreadsheet formulas.

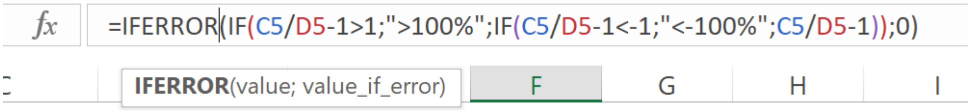
1. All three input methods have limited space to enter and edit a formula compared with IDEs. The space in an IDE makes it possible to see multiple lines of code in a single glance. Because of this, a developer can see a single statement in the context of other statements. Furthermore, they can use horizontal (indentation) or vertical (blank lines) white space to structure their code.

2. In Excel, it is not easy to use white space to structure formulas. Creating vertical white space is possible with in-cell editing and in the formula bar. In both cases, it requires that the user knows the keyboard shortcut for a new line ([alt] + [enter]), and if it is done in the formula bar, the user needs to resize the formula bar manually to create enough space to see the additional blank lines. Horizontal white space is even more difficult. Excel does not accept the tab character in a formula. However,



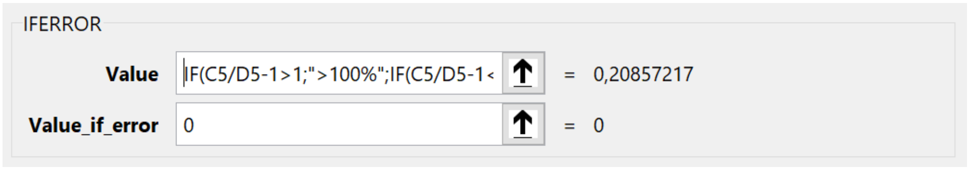
```
=IFERROR(IF(C5/D5-1>1;">100%";IF(C5/D5-1<-1;"<-100%";C5/D5-1));0)
C IFERROR(value; value_if_error)      1.267.395      543.189      43%
```

(a) In-cell editing



```
fx   =IFERROR(IF(C5/D5-1>1;">100%";IF(C5/D5-1<-1;"<-100%";C5/D5-1));0)
C    IFERROR(value; value_if_error)    F    G    H    I
```

(b) Editing in the formula bar



IFERROR

Value	IF(C5/D5-1>1;">100%";IF(C5/D5-1<	= 0,20857217
Value_if_error	0	= 0

(c) Using the function wizard

Figuur 1.2: three ways for entering a formula in Excel

indentation can be created with spaces. If the function wizard is used, Excel will generate the formula automatically, and no white space is added.

3. The limited space available in spreadsheet programs to display formulas in combination with the default behavior that only a single formula can be seen at a glance makes it difficult for spreadsheet users to get an overview of the formulas used in a spreadsheet.

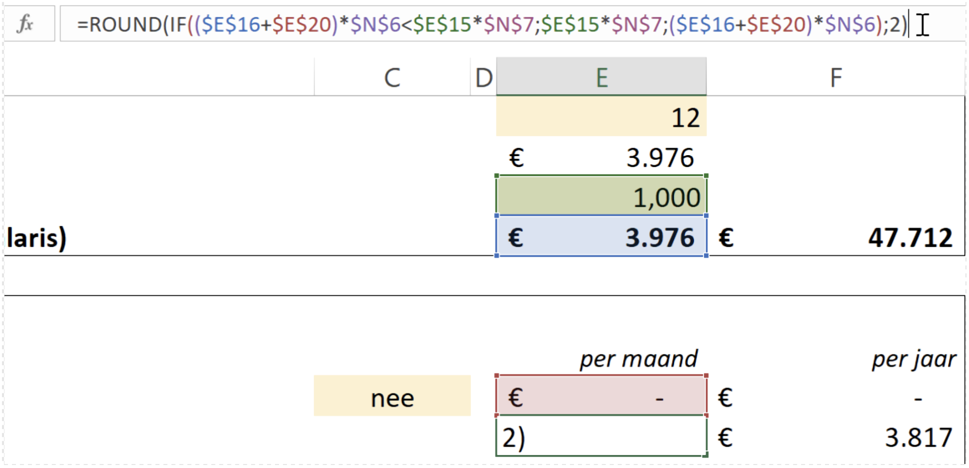
4. Syntax highlighting is not available in spreadsheets. Although color is used in formulas, it is not used to highlight keywords but to highlight references to other cells and for matching parentheses (see Fig. 1.3).

5. Tools for refactoring are lacking, although both Hermans and Dig[15] and Badame and Dig [16] have shown that refactoring for spreadsheets is technically feasible and can support users to create better formulas.

6. Navigation is another area where spreadsheet software is lacking. Relations between formulas are not visible in the user interface. Some tools can visualize dependencies between formulas, but this functionality is limited and does not scale when spreadsheets grow bigger and formulas become more complex.

7. Finally, there is no solution for version management. It is not easy to show the differences between two versions of the same spreadsheet. For example, if a user inserts an empty row in a spreadsheet, all formulas that refer to a cell below this row are automatically changed. Furthermore, because it is common practice to copy formulas down or to the right to simulate loops (see Section 1.2.1), insertion of a single empty row can lead to thousands of changes in formulas. The insertion of the empty

row did not change the working of the spreadsheet model in any way, but the user would be confronted with thousands of changes in formulas. This problem could be circumnavigated by focusing on changes to unique formulas. They can be detected in a spreadsheet by using the R1C1[3] notation instead of the default A1 notation[17]. The thousands of changes can then be reduced to a single change to a unique formula. However, it immediately introduces a new challenge. If two spreadsheets A and B are compared, how does one know which unique formula in spreadsheet A corresponds to the same unique formula in spreadsheet B. Spreadsheet formulas do not possess a unique identifier. The formula's cell address is some form of a unique identifier but can easily be changed when changes are made to the layout of the spreadsheet by inserting or deleting rows or columns. A possible approach for a solution to this challenge is described in Chapter 4.



```
fx  =ROUND(IF(($E$16+$E$20)*$N$6<$E$15*$N$7;$E$15*$N$7;($E$16+$E$20)*$N$6);2)
```

	C	D	E	F
			12	
			€ 3.976	
			1,000	
laris)			**€ 3.976**	**€ 47.712**
			per maand	*per jaar*
	nee		€ -	€ -
			2)	€ 3.817

Figuur 1.3: Cell reference highlighting instead of syntax highlighting

1.2.3 Users

An important difference between software developers and spreadsheet users is that most spreadsheet users did not receive any formal training in software development. They are first, and foremost end-users [18]. There is a growing body of research in which software engineering methods are transferred to the spreadsheet domain to support users in creating better spreadsheets. However, some of these methods expect spreadsheet users to know concepts from the domain of computer science. For example, Erwig introduced ClassSheet [19]. With ClassSheet, it is possible to define a template for spreadsheets that can automatically generate a spreadsheet model. However, it also requires some knowledge of the principles of object-oriented programming. Knowledge most spreadsheet users are lacking. Roy interviewed spreadsheet users about their testing practices [20].

[3]In the R1C1 notation, *R* stands for row and *C* for column with the numbers identifying the row and column number of the cell.

Their answers illustrate that the development of a spreadsheet is often an ad-hoc and organic process. Users are focused on finding a quick solution for their problem and do not think about the spreadsheet structure. Hermans found that spreadsheets have an average lifespan of five years and are used on average by thirteen different users [6]. However, if spreadsheet users are confronted with these findings, they are surprised. They never planned to design a spreadsheet that can be easily maintained for such a lifespan and is easy to use for different users.

1.3 Thesis statement

This thesis aims to find ways in which spreadsheet users can be supported to create spreadsheets that contain fewer errors. As we have seen in the previous section, most spreadsheet challenges concern the user interface of spreadsheet software. Also, it was mentioned that a growing body of research focuses on transferring methods used in software engineering to the domain of spreadsheets. This research includes testing, reverse engineering, code smells, and refactoring. Attention to the user interface of spreadsheets is missing in this research. This thesis will zoom in on the development and maintenance of spreadsheet formulas. Are there ways to improve the interface for creating and maintaining formulas?

We started our research by exploring the idea of using a visual language to create better spreadsheets [21]. It is this central thought that formed the inspiration for the following thesis statement:

A visual language supports professional spreadsheet users in interacting with complex formulas. This results in a reduction in the number of errors made during the creation or maintenance of formulas.

We will use this thesis statement as a guide for our research. It narrows down the scope and provides a roadmap for this dissertation. In the coming chapters, we will examine the different aspects, and based on the conclusions of our research, we will reflect on the statement and answer the question if we think it holds.

1.4 Methodology

To explore the previously mentioned thesis statement, we conducted several different studies.

In these studies, we worked with spreadsheets used in industry. We took inspiration from well-established methods in software engineering and translated them to the domain of spreadsheets.

In the different studies, we used several research methods.

- We used static analysis [22] of formulas to detect code smells in spreadsheet formulas. We analyzed 130 spreadsheets from an industrial partner with the Spreadsheet Scantool developed at Delft University of Technology. The study is reported in Chapter 2.

- We used a controlled experiment [23] to research the effect of delocalized plans in spreadsheets. We asked the participants of the experiment to perform a set of maintenance tasks in a spreadsheet. Each participant had to perform the same tasks, but the spreadsheet differed. See Chapter 3 for a detailed report of this research.

- In Chapter 4, we use a case study [24] to analyze the evolution of two sets of spreadsheets. Within these sets, we analyzed different versions of spreadsheets and discussed our findings with the owners of the spreadsheets in an interview.

- To evaluate users' experiences with our block-based formula language XLBlocks (see Chapter 5 and 6) we used think-aloud studies [25]. The aim of our research in these two chapters is not to prove that XLBlocks is a better formula editor but to understand how it is used and experienced by professional spreadsheet users. Therefore, we specifically chose think-aloud studies because they allowed us to collect as much information as possible about how users experience XLBlocks. In the studies, we asked participants to perform typical spreadsheet tasks. After the tasks, we conducted a semi-structured interview to further inquire about things we noticed during the think-aloud study.

- To evaluate XLBlocks as a programming language, we used the Cognitive Dimensions of Notation (CDN) framework [26]. We asked participants in the studies to evaluate XLBlocks on the different dimensions of the CDN framework.

We are firm believers in open data, and whenever possible, we share our data and source code online. The data is stored following the guidelines of TU Delft, which means, among other things, that it complies with the GDPR and that data is stored and archived for at least ten years for the sake of transparency and audibility. We also value research in an industry setting. Therefore, in some studies, we worked closely with industrial partners. In those cases, the spreadsheets we used were confidential and could not be shared.

1.5 Outline

The chapters of this dissertation are based on peer-reviewed publications in several software engineering conferences. Each chapter is self-contained, and therefore one could notice some repetitions, especially in the introduction of the chapters. The main body of this thesis consists of previously published papers that examine the different aspects of the thesis statement. All chapters are co-authored with Felienne Hermans, and Chapter 4 is also co-authored with Edwin Tazelaar.

The research for this dissertation can be roughly divided into two phases. In phase one, our research aimed to gain a better understanding of how users interact with spreadsheets and what causes the error-proneness. In phase two of the research, we developed and evaluated a visual language to support users in creating and maintaining complex formulas.

1.5.1 Phase I: better understanding of spreadsheets

We will start in Chapter 2 with defining the concept of code smells in spreadsheet formulas. In previous research, Hermans defined code smells for spreadsheets, both on the level of the worksheet and the level of formulas [27], [28]. In these studies, Hermans showed that it is possible to define code smells for spreadsheets and users recognize the different smells. However, the studies did not demonstrate that smells in spreadsheets are bad by definition and will cause problems. This chapter explores the hypothesis that spreadsheets containing fewer code smells are less error-prone, easier to understand, and easier to maintain. We analyzed a set of 54 pairs of spreadsheets that we obtained from financial modeling company F1F9. Each pair consists of a spreadsheet created by one of F1F9's customers and a corresponding model with the same functionality as the first model but rebuilt from scratch by professional spreadsheet modelers of F1F9 using the FAST standard. We scanned all spreadsheets for smells with the Spreadsheet Scantool developed at Delft University of Technology [28]. We hypothesized that the spreadsheets created by the professional spreadsheet developers of F1F9 should contain less smelly formulas than the spreadsheets built by their customers. The results showed that the F1F9 spreadsheets suffer from smells to a much lower extent than the customer's sheets. This chapter previously appeared in the Proceedings of the 2015 International Conference on Software Maintenance and Evolution (ICSME) [29].

In Chapter 3, we explore the effect of delocalized plans in spreadsheets on the comprehensibility of spreadsheets. In this study, we conducted a controlled experiment. One hundred seven participants were asked to perform several comprehension tasks in a spreadsheet. For this experiment, we created two versions of the same spreadsheet. In one spreadsheet, the data was structured so that all data needed for a particular formula was grouped closely together. In the other spreadsheet, similar data were grouped, and as a result, the data needed for a single formula could be spread all over the spreadsheet. The participants were randomly assigned to one of the two models. The results reveal that participants perform significantly better when they perform the comprehension tasks on the model where all data needed for a single calculation was grouped closely together. The absence of delocalized plans in the spreadsheet led to better comprehensibility. This chapter was published in the Proceedings of the 25th (2017) International Conference on Program Comprehension (ICPC) [30].

Chapter 4 describes a case study of the evolution of a set of spreadsheets. As mentioned in Section 1.2.3, spreadsheets have an average life span of 5 years. During this life span, it is to be expected that the model needs maintenance or that data is updated. The update of data and the maintenance of formulas are moments when errors can be introduced in the model. In this chapter, we describe two case studies on two different sets of spreadsheets. We follow the evolution of these spreadsheets over a period of three years. We needed to develop an algorithm to detect and visualize the changes made during this period. Results of the case study indicate that studying the evolution of a spreadsheet helps users to identify areas in the spreadsheet that are error-prone, likely to change, or that could benefit from refactoring. This work appeared in the Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME) [31].

1.5.2 Phase II: development of a visual language for spreadsheet formulas

In Chapter 5, we finally turn our attention to the user interface of spreadsheets. As we have learned from the concept of code smells in spreadsheets (see Chapter 2) and research on spreadsheet errors[11], we know that most spreadsheet errors have their origin in formulas. In Section 1.2.2 we also saw that the interface for creating and editing formulas is challenging in itself. Therefore, we wondered if improving the interface would help to improve the overall quality of spreadsheets. Given the success of block-based languages in studies on the performance of novice programmers, we hypothesized that a block-based formula editor could similarly support spreadsheet users.

Therefore, we developed XLBlocks, a block-based formula language for spreadsheets. To evaluate XLBlocks, we conducted a think-aloud study with thirteen experienced spreadsheet users. In the study, we asked them to create and edit several spreadsheet formulas with XLBlocks. After the think-aloud study, participants evaluated XLBlocs against the Cognitive Dimensions of Notations framework. This evaluation revealed that XLBlocks received, on all dimensions, a better evaluation than the default text-based formula editor of Excel. The think-aloud study showed that users were supported in thinking about the formula they were constructing because of the drag and drop interface. They experienced, in comparison with the text-based formula editor, more freedom in the sequence in which they could construct the formula. The study also indicated areas where XLBlocks could be further improved. Better use of color could help make the formulas easier to read, an improved type-checking system should prevent users from connecting invalid combinations of blocks, and in this version of XLBlocks, only new formulas could be generated. Users indicated that it would be beneficial to generate a block-based model of an existing formula. This chapter was published in the Proceedings of the 2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) [32].

Chapter 6 presents a second study with XLBlocks. In this study, we focus on the effect of a block-based language on formula comprehension in spreadsheets. Research has shown that block-based languages positively influence the comprehension of code [33]. We hypothesize that the same goes for spreadsheets and that a block-based formula language will positively affect formula comprehension. To study this, we extended XLBlocks with the functionality to generate a block-based representation of an existing formula. Next, we conducted a think-aloud study with twenty-one experienced spreadsheet users. They were asked to perform a set of twelve comprehension tasks on an existing spreadsheet. After the think-aloud study, we interviewed the participants and asked them to reflect on the use of XLBlocks. We used again the Cognitive Dimensions of Notations framework to structure the interview. Results of the study showed that the participants preferred the block-based representation of the formula over the textual representation when analyzing or explaining formulas. They also preferred XLBlocks for implementing non-trivial changes. Furthermore, they indicated that the presence of named parameters and the absence of parentheses and commas made the formulas easier to understand. Also, the visualization enabled them to separate smaller parts in the formula, which improved comprehension. Finally, the possibilities to easily navigate from formula to formula made it easier to understand how formulas were related and gave a better understanding of the spreadsheet as a whole. This chapter appeared in the Proceedings of the 29th (2021) International Conference on Program Comprehension (ICPC) [34].

Finally, Chapter 7 presents the conclusions of this dissertation and discusses future work.

2 Code Smells in Spreadsheet Formulas Revisited on an Industrial Dataset

In previous work, code smells have been adapted to be applicable on spreadsheet formulas. The smell detection algorithm used in this earlier study was validated on a small dataset of industrial spreadsheets by interviewing the users of these spreadsheets and asking them about their opinion about the found smells. In this paper a more in depth validation of the algorithm is done by analyzing a set of spreadsheets of which users indicated whether or not they are smelly.

This new dataset gives us the unique possibility to get more insight in how we can distinguish 'bad' spreadsheets from 'good' spreadsheets. We do that in two ways: For both the smelly and non smelly spreadsheets we 1) have calculated the metrics that detect the smells and 2) have calculated metrics with respect to size, level of coupling, and the use of functions. The results show that indeed the metrics for the smells decrease in spreadsheets that are not smelly. With respect to size we found to our surprise that the improved spreadsheets were not smaller, but bigger. With regard to coupling and the use of functions both datasets are similar. It indicates that it is difficult to use metrics with respect to size, degree of coupling or use of functions to draw conclusions on the complexity of a spreadsheet.

This chapter has been published as *B. Jansen and F. Hermans. Code Smells in Spreadsheet Formulas Revisited on an Industrial Dataset, ICMSE'15* [29]

Spreadsheets could be considered the most successful end-user programming platform, with an estimated 55 million people using them in the US alone. Because spreadsheets are so widely used as programming tools, it is plausible to apply methods from software engineering to them in order to improve them. This has been done in previous work, among others by Hermans *et al.* [27] who translated some of Fowler's code smells [35] to the realm of spreadsheets.

In their paper, a method for the detection of *spreadsheet smells* is described, including for example a long list of referenced cells or deeply nested conditional formulas. To evaluate their smell detecting algorithm, Hermans *et al.* detected smells in ten spreadsheets created by employees of an investment bank, and subsequently interviewed the users of these spreadsheets, asking their opinion about the found smells. This study had two obvious limitations: firstly, the dataset used in this evaluation was very small, consisting of only 10 spreadsheets stemming from one company. Secondly, it was not known in advance if these spreadsheet were suffering from smells; users were just asked for 'complex spreadsheets'.

This paper presents a more extensive investigation of spreadsheet smells on an entirely new dataset, obtained from financial modeling company F1F9. Employees of F1F9 develop financial models in Excel for customers, often based upon the customer's existing spreadsheet models. We have obtained 54 pairs of spreadsheets consisting of the original model developed by the customer and the rebuilt model created by F1F9 employees. Customers in general reach out to F1F9 because they cannot maintain their spreadsheets models anymore, in other words: they are smelly. As such, these pairs of smelly and non-smelly versions of the same spreadsheet provide ample opportunity for us to investigate what characterizes a smelly spreadsheet.

To do so, we have performed an evaluation in which we detected smells for both smelly and non-smelly spreadsheets. We have applied both the Wilcoxon Signed-Ranks Test for Paired Samples and the Wilcoxon-Mann-Whitney test to see if there is significant difference between the two types of spreadsheets. We find that indeed the rebuilt spreadsheets contain smells less frequently. In addition to calculating smells, we also calculated size and coupling metrics of the two types of spreadsheets and investigated their use of functions. Surprisingly enough, the rebuilt sheets are not smaller, but bigger, and seem very similar in terms of coupling and function use. Hence, these metrics do not offer value when trying to distinguish maintainable from smelly spreadsheets.

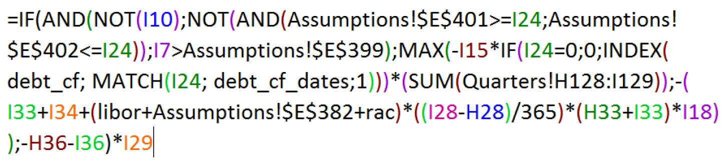
With our work, we improve upon the existing, preliminary, study in two ways. Firstly, our dataset is bigger, consisting of 108 spreadsheets. More importantly, our set is based on pairs of spreadsheets, one being an original, smelly spreadsheet, and the other being a rebuilt, well-structured model, allowing us to pair-wise compare the models.

The remainder of this paper is structured as follows: in the next section we give background information about the smells that we used to analyze the spreadsheets. In Section 2.2 we describe the setup of our analysis. We explain the content of the dataset and the procedure we followed to calculate the different metrics. The results of the analysis are presented in section 2.3. In Section 2.4 we discuss the results in more detail and put them in the context of the FAST standard that was used by F1F9 to rebuild the financial models. Several issues that affect the applicability and suitability of the findings are discussed in section 2.5 and we finish the paper with related work (Section 2.6) and the concluding remarks (2.7).

2.1 Background

Hermans *et. al.* [27] introduced 5 smells in spreadsheet formulas:

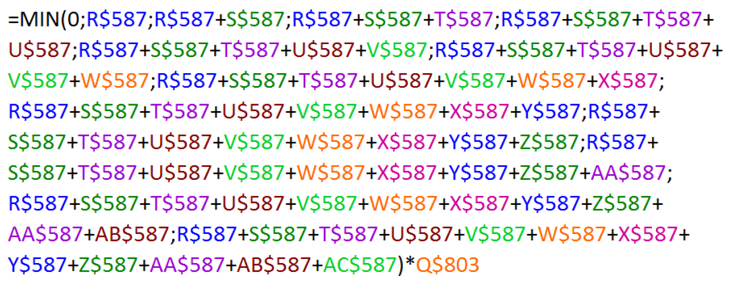
- **Multiple Operations**: Inspired by the code smell Long Method, this smell indicates the length of the formula. It measures the total number of operations that a formula contains. Figure 2.1 shows a formula that is suffering from this smell with a total of 15 unique operations within the formula.



```
=IF(AND(NOT(I10);NOT(AND(Assumptions!$E$401>=I24;Assumptions!
$E$402<=I24));I7>Assumptions!$E$399);MAX(-I15*IF(I24=0;0;INDEX(
debt_cf; MATCH(I24; debt_cf_dates;1)))*(SUM(Quarters!H128:I129));-(
I33+I34+(libor+Assumptions!$E$382+rac)*((I28-H28)/365)*(H33+I33)*I18)
);-H36-I36)*I29
```

Figuur 2.1: Example of Multiple Operations Smell

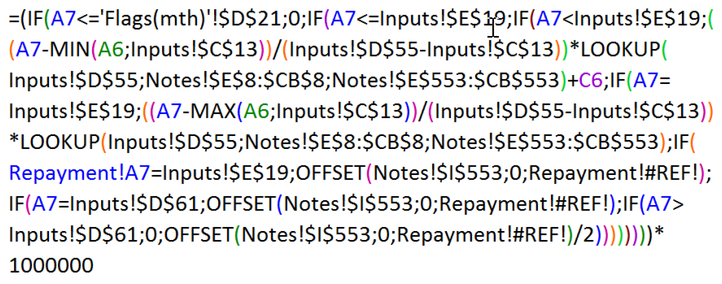
- **Multiple References**: Another well known code smell is Many Parameters. The spreadsheet formula equivalent is Multiple References. It counts the number of ranges a formula is referring to. An example of this smell is a formula with 79 references that is shown in Figure 2.2.



```
=MIN(0;R$587;R$587+S$587;R$587+S$587+T$587;R$587+S$587+T$587+
U$587;R$587+S$587+T$587+U$587+V$587;R$587+S$587+T$587+U$587+
V$587+W$587;R$587+S$587+T$587+U$587+V$587+W$587+X$587;
R$587+S$587+T$587+U$587+V$587+W$587+X$587+Y$587;R$587+
S$587+T$587+U$587+V$587+W$587+X$587+Y$587+Z$587;R$587+
S$587+T$587+U$587+V$587+W$587+X$587+Y$587+Z$587+AA$587;
R$587+S$587+T$587+U$587+V$587+W$587+X$587+Y$587+Z$587+
AA$587+AB$587;R$587+S$587+T$587+U$587+V$587+W$587+X$587+
Y$587+Z$587+AA$587+AB$587+AC$587)*Q$803
```

Figuur 2.2: Example of Multiple References Smell

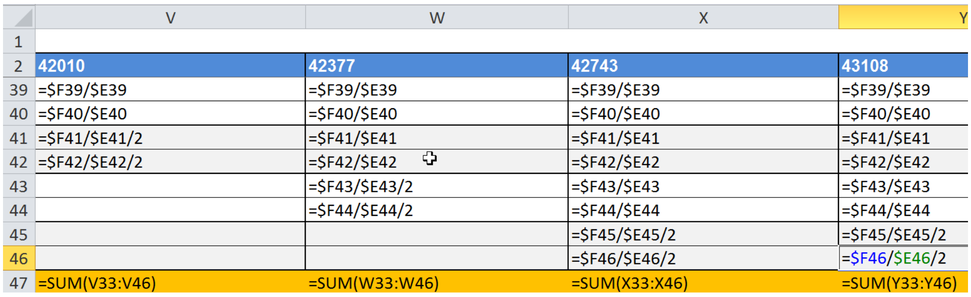
- **Conditional Complexity**: Many nested conditional operations are considered as a threat to code readability [35]. The same is true for spreadsheet formulas. The Conditional Complexity smell measures the number of conditionals contained by a formula. Figure 2.3 shows a formula with 7 nested IFs functions. This was the maximum number of nested IFs that was allowed up to Excel 2003.
- **Long Calculation Chain**: In spreadsheets, it is common that formulas refer to other formulas. Therefore, one could say that a spreadsheet consists of a collection of calculation chains. Tracing a long calculation chain is considered by users as a tedious task. The smell is measured by the length of the longest path of cells that need to be referenced when computing the value of the formula.



```
=(IF(A7<='Flags(mth)'!$D$21;0;IF(A7<=Inputs!$E$19;IF(A7<Inputs!$E$19;(
(A7-MIN(A6;Inputs!$C$13))/(Inputs!$D$55-Inputs!$C$13))*LOOKUP(
Inputs!$D$55;Notes!$E$8:$CB$8;Notes!$E$553:$CB$553)+C6;IF(A7=
Inputs!$E$19;((A7-MAX(A6;Inputs!$C$13))/(Inputs!$D$55-Inputs!$C$13))
*LOOKUP(Inputs!$D$55;Notes!$E$8:$CB$8;Notes!$E$553:$CB$553);IF(
Repayment!A7=Inputs!$E$19;OFFSET(Notes!$I$553;0;Repayment!#REF!);
IF(A7=Inputs!$D$61;OFFSET(Notes!$I$553;0;Repayment!#REF!);IF(A7>
Inputs!$D$61;0;OFFSET(Notes!$I$553;0;Repayment!#REF!)/2)))))))*
1000000
```

Figuur 2.3: Example of Conditional Complexity Smell

- **Duplicated Formulas** The equivalent of the Duplicate Code smell in spreadsheets is the *Duplicated Formulas* smell. The smell as described in [27] occurs at formulas that are partially the same as others. The smell is measured by the number of formulas, located in the same worksheet and expressed in relative R1C1 notation, with which a formula shares at least one proper subtree. In the original study, it was found that users found this smell hard to understand. Therefore we changed the definition for this smell. It is now measured by the number of identical formulas that are located in the same worksheet and having at least one function or operator. Row 39 in Figure 2.4 shows an example of four identical formulas.



	V	W	X	Y
1				
2	42010	42377	42743	43108
39	=$F39/$E39	=$F39/$E39	=$F39/$E39	=$F39/$E39
40	=$F40/$E40	=$F40/$E40	=$F40/$E40	=$F40/$E40
41	=$F41/$E41/2	=$F41/$E41	=$F41/$E41	=$F41/$E41
42	=$F42/$E42/2	=$F42/$E42	=$F42/$E42	=$F42/$E42
43		=$F43/$E43/2	=$F43/$E43	=$F43/$E43
44		=$F44/$E44/2	=$F44/$E44	=$F44/$E44
45			=$F45/$E45/2	=$F45/$E45/2
46			=$F46/$E46/2	=$F46/$E46/2
47	=SUM(V33:V46)	=SUM(W33:W46)	=SUM(X33:X46)	=SUM(Y33:Y46)

Figuur 2.4: Example of Duplicated Formula Smell

We received the dataset that we use in this paper from F1F9[1]. They are the world largest financial model building firm. They build their models using spreadsheets, according to the FAST standard. The FAST standard [36] was first developed by employees of F1F9. It is now maintained by the FAST Standard Organization[2]. The standard is primarily concerned with good spreadsheet design. Its acronym stands for Flexible, Appropriate, Structured, and Transparent. It aims to support spreadsheet designers to build spreadsheets that: Are

[1]To protect the confidentiality of the models we only had access to the dataset on F1F9's premises and then only indirectly whereby our software automatically generated and stored only the necessary survey statistics. At no point did we have direct access to the models, nor did our software extract any commercial data from the models.

[2]http://www.fast-standard.org/

free of fundamental omissions; Have no logical errors; Can be created under short lead times; Can be easily used and reviewed; Are readily adaptable when circumstances change.

2.2 Experimental Setup

The dataset we use for this paper consists of 54 pairs of spreadsheets. For every pair one spreadsheet was created by a client of F1F9, the other one an improved version built by consultants of F1F9 according to the FAST standard. Both spreadsheets have the same functionality and deliver the same results for identical input. However, the models within the spreadsheets are completely different. F1F9 built their version completely from scratch. All spreadsheets contain financial models.

When we received the dataset, it consisted of a total of 146 spreadsheets. However, we discovered that the set contained some duplicates and that for some client files the matching F1F9 file was missing and vice versa. So after an initial cleaning, a set of 130 files remained. Subsequently, we analyzed these spreadsheets with the Spreadsheet Scantool, developed at Delft University of Technology. The tool runs on the previously developed Breviz core that was made for spreadsheet visualization and smell detection [37]. Some of the remaining files were password protected, corrupt or otherwise unreadable by the scantool and were therefore excluded from the dataset. Of course if a client file was unreadable, we also had to exclude the matching F1F9 file. Eventually we ended up with 108 scanned files.

The spreadsheets in the Client set were perceived by their users as problematic. It was because of this reason, that they asked F1F9 to rebuild these models. What makes the dataset interesting for our research is that we have one set of spreadsheets that are perceived by their users as problematic and a matching set of spreadsheets that, according to professional model builders, are easier to understand, less error-prone and less difficult to maintain.

In earlier studies regarding smells in spreadsheets the smells were validated by either asking users about their opinion about the smelliness [27] or by manual inspecting the detected smells to see if they were actually smelly [38]. What was missing in these validations, was ground truth about the smelliness of a spreadsheet. Fortunately, we can solve this now with the above mentioned dataset.

One would expect that a spreadsheet that is considered easier to understand, less error-prone and less difficult to maintain contains less smells than a spreadsheet that is perceived as problematic. This brings us to the main question for this research:

R1 Do spreadsheets that are perceived as easier to understand, contain fewer smelly cells than spreadsheets that are perceived as problematic?

In earlier work [39], metrics for size, coupling and the use of functions were used to characterize spreadsheets in large corpora. We will use these metrics to analyze the differences between the client and the F1F9 spreadsheets. Table 2.1 gives an overview of these metrics.

Most of these metrics are self-explanatory, however a few deserve some further explanation [39]:

Tabel 2.1: Overview of Metrics

Dimension	Metric
Size	s1 # non-empty cells per spreadsheet
	s2 # worksheets per spreadsheet
	s3 # formulas per spreadsheet
	s4 # unique formulas per spreadsheet
	s5 length of formula in characters (measured per cell)
Coupling	c1 % external links per spreadsheet
	c2 # interworksheet connections per spreadsheet
	c3 path depth per formula
	c4 total number of transitive precedents per formula
Use of functions	f1 Number of unique functions per formula
	f2 Parse three depth per formula
	f3 Number of preceding cells per formula

- **Number of unique formulas per spreadsheet (s4)**: It is common practice in spreadsheets to define a formula in once cell and then copy it down or right to other cells. As a consequence, many of the formula cells in a spreadsheet contain the same formula except for the references to the other cells. Therefore, we also measure the number of unique formulas in the spreadsheet. We determine the unique formulas by looking at the relative R1C1 notation of the formula. As described by Sajaniemi [17], this notation stays the same even if you copy a formula down or right.

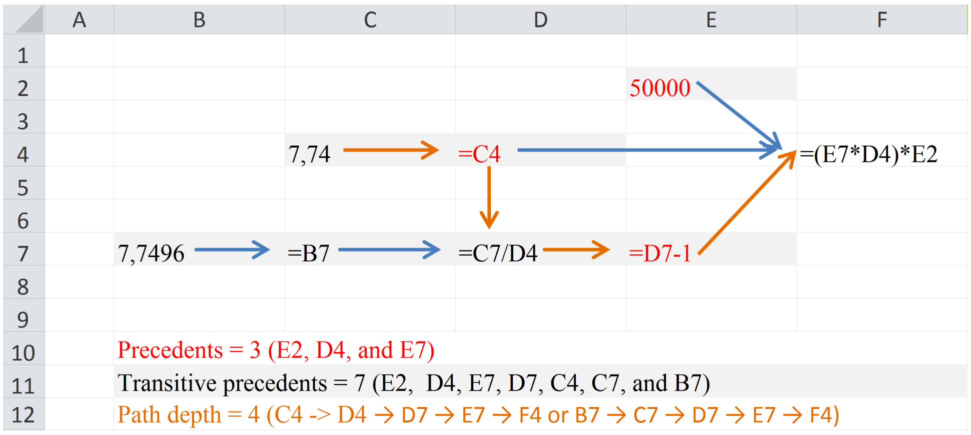
- **Path depth (c3), transitive precedents (c4), and number of preceding cells (f3)**: In most cases formulas receive input from other cells. This is what we measure with the number of preceding cells per formula. However, these precedents could be formulas themselves that, in turn, have their own precedents. The number of transitive precedents is calculated by tracing along these precedents until on all branches a cell is reached without any precedents. The path depth is the longest calculation chain within the tree of precedents. See also Figure 2.5.

- **Parse tree depth (f2)**: This metric indicates how nested a formula is. The formula A1 + A2 has a parse tree depth of 2, the formula (A1 - A2) / (A3 * SQRT(A5)) a parse tree depth of 5.

Calculating these metrics for the spreadsheets will enable us to answer the second research question:

R2 What are the differences with respect to size, level of coupling and the use of functions between spreadsheets that are perceived as easier to understand and spreadsheets that are perceived as problematic?

To answer both research questions, we calculated for each file the metrics that indicate one of the five smells that were described at the beginning of this chapter and the metrics with respect to size, coupling, and use of functions.



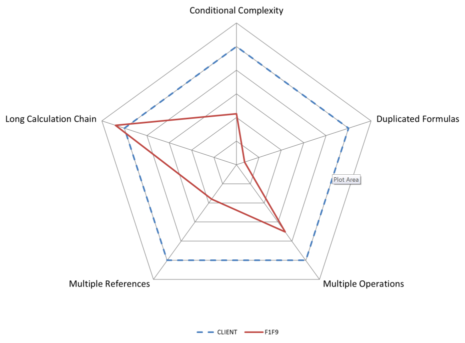
	A	B	C	D	E	F
1						
2					50000	
3						
4			7,74	=C4		=(E7*D4)*E2
5						
6						
7		7,7496	=B7	=C7/D4	=D7-1	
8						
9						
10		Precedents = 3 (E2, D4, and E7)				
11		Transitive precedents = 7 (E2, D4, E7, D7, C4, C7, and B7)				
12		Path depth = 4 (C4 -> D4 → D7 → E7 → F4 or B7 → C7 → D7 → E7 → F4)				

Figuur 2.5: Precedents, Transitive Precedents, and Path Depth ([39])

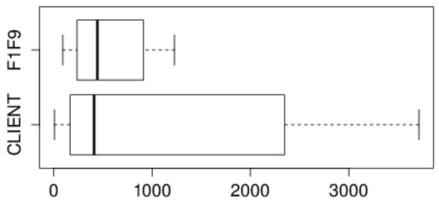
2.3 Results

2.3.1 Smells

Figure 2.6 displays the results for the smell metrics. We use the radar chart to visualize all metrics in a single figure. The chart shows the relative score for each smell. The F1F9 scores (red line) are represented as a percentage of the Client scores (blue line) (Client = 100%).



Figuur 2.6: Relative score of Smells



Figuur 2.7: F1F9 has a slightly higher median, but a much smaller interquartile distance

For the exact figures see Table 2.2. It shows for each smell the median of the number of times the smell occurred in each spreadsheet. The last column gives the score for the F1F9 spreadsheets as a percentage of the score of the Client sheets and were used in Figure 2.6.

Tabel 2.2: Overview of Smells

Metric	Client	F1F9	F1F9 (%)
Multiple operations	101.0	71.0	70.3%
Multiple references	136.5	49.0	35.9%
Conditional complexity	36.0	15.5	43.1%
Long calculation chain	412.0	444.5	107.8%
Duplicated formulas	296.0	21.5	7.2%

We can see that overall the number of occurrences of the smells decrease. We observe a dramatic decrease of the occurrence of the *Duplicated formula* smell and also see a clear difference for the *Multiple references* and *Conditional complexity* smells. In Section 2.4, we will explain some possible causes for these findings. *Long calculation chain* forms an exception because the number of occurrences of this smell is slightly higher in the F1F9 sheets. We have analyzed this in more detail. Figure 2.7 shows the boxplot for this smell for both the Client and the F1F9 sheets. The boxplot displays the minimum, 1st quartile, median, 3rd quartile, and maximum value. In the Client data set, there is one spreadsheet with a calculation chain of 9,995 cells, that can be considered as an outlier. We have excluded it from the boxplot because we wanted to visualize the difference in interquartile distance between F1F9 and the Client, which is not affected by the outlier. It shows that although the median for the number of occurrences of the *Long calculation* smell is slightly higher for F1F9 than the Client, the 3rd quartile and maximum value were decreased dramatically. There are fewer spreadsheets that suffer in a high degree from the *Long calculation* smell in the F1F9 dataset.

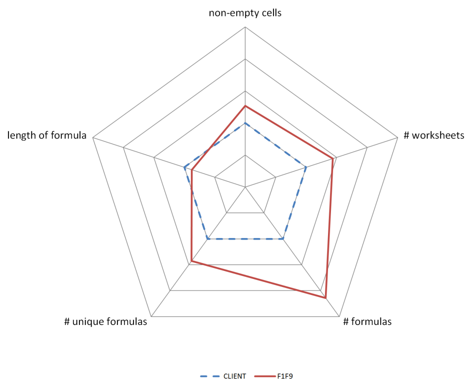
2.3.2 Size, Coupling and Use of Functions

The metrics for the dimension size have been summarized in Figure 2.8 and the exact figures can be found in Table 2.3. It turns out that almost every size metric has increased for the F1F9 spreadsheets. Only the length of the formulas decreases as compared to the Client sheets. Notable is also the number of formulas. This metric has increased much more than the other size metrics.

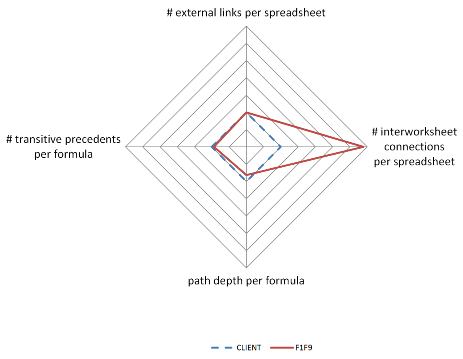
Tabel 2.3: Overview of Size Metrics

Metric	Client	F1F9	F1F9 (%)
s1 # non-empty cells	170,202.0	215,998.5	126.9%
s2 # worksheets	19.5	28.0	143.6%
s3 # formulas	92,954.5	198,711.0	213.8%
s4 # unique formulas	1,467.0	2,094.0	142.7%
s5 formula length	32.0	28.0	87.5%

To measure the level of coupling, we have analyzed both the external (to other spreadsheets) and internal (within the same spreadsheet) links between worksheets, the path depth per formula and the total number of transitive precedents of a formula. The results of this analysis are visualized in Figure 2.9 and the exact figures summarized in Table 2.4.



Figuur 2.8: Relative score on dimensions of size



Figuur 2.9: Relative score on dimensions of coupling

Tabel 2.4: Overview of Coupling Metrics

Metric	Client	F1F9	F1F9 (%)
c1 # external links	0.0	0.0	100.0%
c2 # interworksheet connections	60.5	205.0	338.8%
c3 Path depth	16.0	13.0	81.3%
c4 # transitive precedents	135.0	127.0	94.1%

From the results, it seems that both datasets are almost identical on the degree of coupling, except for the number of interworksheet connections. These are much higher within the F1F9 dataset.

Finally, we have analyzed the use of functions in both datasets, by looking at the number of unique functions used, the parse tree depth, and the number of preceding cells per formula. Figure 2.10 summarizes the results for these metrics. Exact figures can be found in Table 2.5.

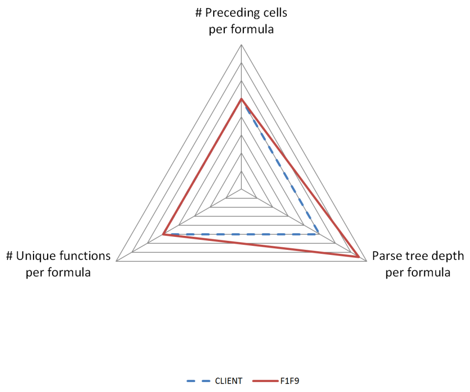
Tabel 2.5: Metrics on the Use of Functions

Metric	Client	F1F9	F1F9 (%)
f1 # unique functions	1	1	100.%
f2 parse tree depth	2	3	150%
f3 # preceding cells	3	3	100%

With respect to the use of functions, both datasets are very similar. The only difference is the median for the parse tree depth, which is 3 for F1F9 and 2 for the Client sheets.

2.3.3 Significance of the Differences

These results give us an indication of the differences between smelly and non-smelly spreadsheets. However, we do not know yet if these differences are statistically signi-



Figuur 2.10: Relative score on Use of Functions

ficant. Because we have pairs of smelly and non-smelly spreadsheets we can do a paired group comparison to determine if there is a significant difference. To do so, we used the Wilcoxon Signed-Ranks Test for Paired Samples. However, we can do this test only for the metrics on spreadsheet level (because we have pairs of spreadsheets). If the metric is a characteristic of a formula, a paired comparison is not possible. We do not have pairs of formulas that we can compare. For these metrics we have to test if the distribution of the two datasets is different. We calculated that using the Wilcoxon-Mann-Whitney test. Both tests give us a p-value that can be found in Table 2.6. We have denoted the metrics on spreadsheet level with an 's' and the metrics that are characteristics of formulas with an 'f'.

If there was a significant difference, we have calculated the effect with the Cliff's Delta d. For both metric s2 (number of Worksheets) and metric c2 (number of interworksheet connections) the effect is large ($d \geq 0.47$). For *Duplicated formulas*, metric c1 (number of external links), and metric c4 (number of transitive precedents) the effect is medium ($0.33 \leq d < 0.47$). For *Conditional complexity*, *Multiple references*, metric s1 (number of non-empty cells), metric s3 (number of formulas) the effect is small ($0.147 \leq d < 0.33$). For the other metrics the effect is negligible.

Tabel 2.6: Statistical Analysis of Client and F1F9 Datasets

Dimension	Metric	Level	p-value	d
Smells	Conditional complexity	s	<0.01	0.158
	Duplicated formulas	s	<0.01	0.412
	Multiple Operations	s	<0.01	0.030
	Multiple References	s	<0.01	0.310
	Long calculation chain	s	<0.05	0.055
Size	s1 # non-empty cells	s	<0.01	0.228
	s2 # worksheets	s	<0.01	0.536
	s3 # formulas	s	<0.01	0.257
	s4 # unique formulas	s	>0.05	-
	s5 formula length	f	<0.01	0.078
Coupling	c1 # external links	s	<0.01	0.400
	c2 # interworksheet conn.	s	<0.01	0.835
	c3 path depth	f	<0.01	0.037
	c4 # transitive precedents	f	<0.01	0.379
Use of Functions	f1 # preceding cells	f	<0.01	0.071
	f2 parse Tree Depth	f	<0.01	0.096
	f3 # unique functions	f	<0.01	0.023

2.4 Interpretation

In the previous section, we have described the results of our analysis. We found that the F1F9 sheets were less smelly, but, to our surprise, also bigger. With regards to coupling and the way functions were used, the sets appear more similar. In this section, we will further discuss some of these results. Although we were not able to interview the users of the spreadsheets, we assume that they perceived the F1F9 spreadsheets as easier to maintain and less error-prone. If this was not the case, F1F9 would have gone out of business a long time ago. But what is causing the difference between the F1F9 and the Client spreadsheets? Of course the employees of F1F9 build complex spreadsheet models for a living, but maybe even more importantly they make use of the FAST standard to build these models. A further explanation of some of the concepts and terminology of the FAST standard will help us to better understand the found results.

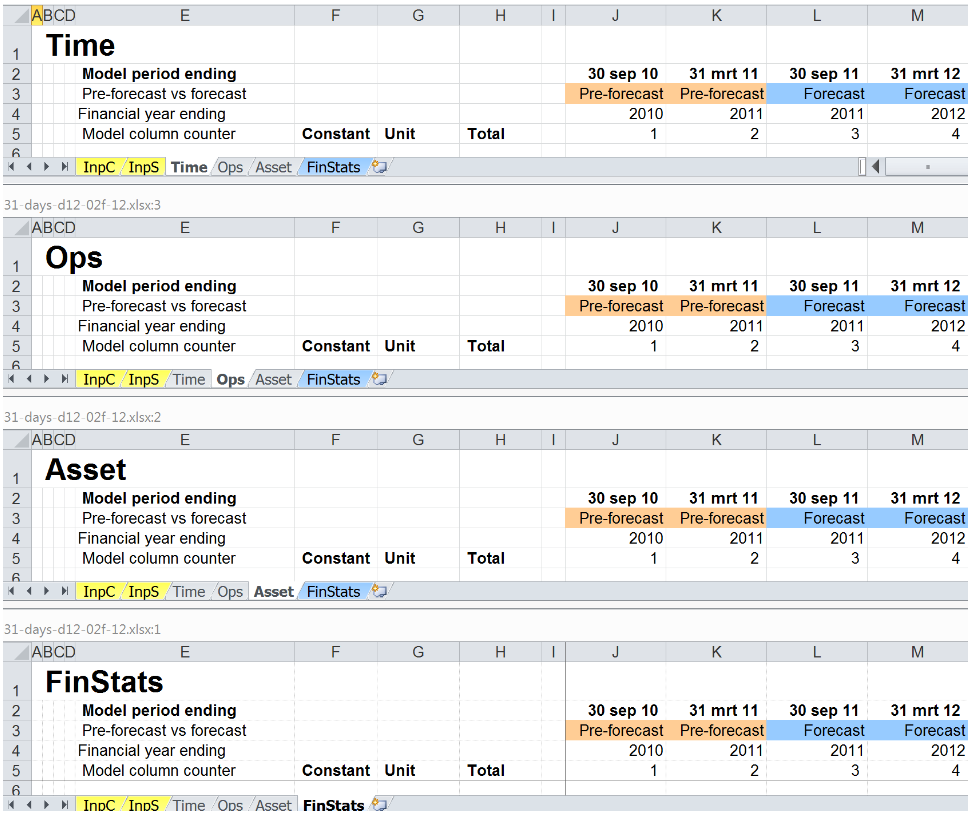
The FAST standard divides a spreadsheet in different hierarchical levels to organize their guidelines. The highest level is the workbook itself, followed by the individual worksheets. According to the FAST standard there can be many worksheets within a workbook but each worksheet always fits in one of the following four functional classes:

1. **Foundation**: The basis for the financial model. These worksheets contain all the inputs, timing rules, assumptions and indexation.
2. **Workings**: The engine of the model. All calculations necessary for the final result of the model are made on these sheets.
3. **Presentation**: The output of the model, usually made up of financial statements, charts and summaries. These sheets are used for decision making by the users of

the model.

4. **Control**: These sheets assist the builder during the process of creating the model. It normally contains list of pending changes, version control, table of contents, error checking, etc. Furthermore, if scenario planning or sensitivity analysis is used, they are controlled from this sheet.

Figure 2.11 shows several worksheets of a financial model that was built according to the FAST standard. The sheets InpC, INpS, and Time are examples of a Foundation sheet; Ops, Asset, and Finstats are examples of a Workings sheet.



	ABCD	E	F	G	H	I	J	K	L	M
1	Time									
2		Model period ending					30 sep 10	31 mrt 11	30 sep 11	31 mrt 12
3		Pre-forecast vs forecast					Pre-forecast	Pre-forecast	Forecast	Forecast
4		Financial year ending					2010	2011	2011	2012
5		Model column counter	Constant	Unit	Total		1	2	3	4
6										

InpC InpS Time Ops Asset FinStats

31-days-d12-02f-12.xlsx:3

	ABCD	E	F	G	H	I	J	K	L	M
1	Ops									
2		Model period ending					30 sep 10	31 mrt 11	30 sep 11	31 mrt 12
3		Pre-forecast vs forecast					Pre-forecast	Pre-forecast	Forecast	Forecast
4		Financial year ending					2010	2011	2011	2012
5		Model column counter	Constant	Unit	Total		1	2	3	4
6										

InpC InpS Time Ops Asset FinStats

31-days-d12-02f-12.xlsx:2

	ABCD	E	F	G	H	I	J	K	L	M
1	Asset									
2		Model period ending					30 sep 10	31 mrt 11	30 sep 11	31 mrt 12
3		Pre-forecast vs forecast					Pre-forecast	Pre-forecast	Forecast	Forecast
4		Financial year ending					2010	2011	2011	2012
5		Model column counter	Constant	Unit	Total		1	2	3	4
6										

InpC InpS Time Ops Asset FinStats

31-days-d12-02f-12.xlsx:1

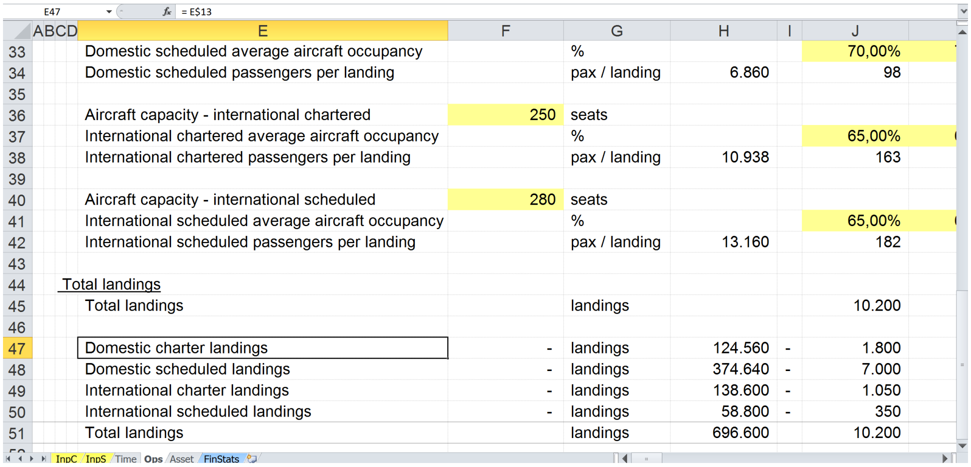
	ABCD	E	F	G	H	I	J	K	L	M
1	FinStats									
2		Model period ending					30 sep 10	31 mrt 11	30 sep 11	31 mrt 12
3		Pre-forecast vs forecast					Pre-forecast	Pre-forecast	Forecast	Forecast
4		Financial year ending					2010	2011	2011	2012
5		Model column counter	Constant	Unit	Total		1	2	3	4
6										

InpC InpS Time Ops Asset FinStats

Figuur 2.11: An example of a consistent column structure that has been maintained across all sheets

In the FAST standard a worksheet is divided in several calculation blocks. A calculation block can be considered as an autonomous paragraph on a worksheet and is always responsible for a single calculation. Rows 44 through 51 in Figure 2.12 show an example of a calculation block. The calculation block itself consists of the ingredients for the calculation (row 47: Domestic charter landings trough 50: International scheduled landings) and the actual calculation (row 51: Total landings).

The lowest level of a financial model is formed by the individual line items (for example row 49 'International charter landings' in Figure 2.12). It's defined as a unit of information displayed on a row or column, of its own with its own label.



E47 = E$13

	ABCD	E	F	G	H	I	J
33		Domestic scheduled average aircraft occupancy		%			70,00%
34		Domestic scheduled passengers per landing		pax / landing	6.860		98
35							
36		Aircraft capacity - international chartered	250	seats			
37		International chartered average aircraft occupancy		%			65,00%
38		International chartered passengers per landing		pax / landing	10.938		163
39							
40		Aircraft capacity - international scheduled	280	seats			
41		International scheduled average aircraft occupancy		%			65,00%
42		International scheduled passengers per landing		pax / landing	13.160		182
43							
44	Total landings						
45		Total landings		landings			10.200
46							
47		Domestic charter landings	-	landings	124.560	-	1.800
48		Domestic scheduled landings	-	landings	374.640	-	7.000
49		International charter landings	-	landings	138.600	-	1.050
50		International scheduled landings	-	landings	58.800	-	350
51		Total landings		landings	696.600		10.200

InpC InpS Time Ops Asset FinStats

Figuur 2.12: Screenshot of a FAST model that shows an example of a calculation block

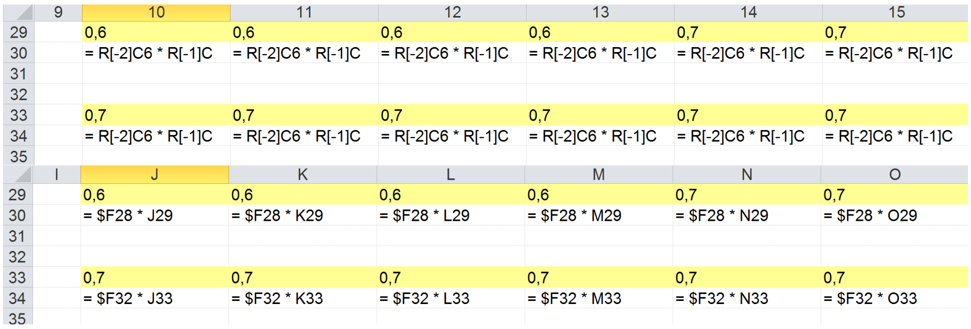
In the remainder of this section, we will analyze the possible effects of the FAST guidelines on the metrics and smells we have calculated for the F1F9 sheets. We will first discuss the smells and at the end of the section focus on the metrics for size, coupling, and the use of functions.

2.4.1 Smells

First of all FAST strongly advises to create short and easy to understand formulas. This explains why we find less formulas that suffer from the *Multiple Operations* and *Multiple References* smells. Furthermore, the standard discourages the use of IF and even prohibit the use of nested IFs. We see that reflected in the lower occurrence of the *Conditional Complexity* Smell.

FAST also dictates that a calculation should only be made once. If the result of the calculation is needed somewhere else in the model, one should link back to the original calculation. Furthermore, formulas should be consistent along the row or column axis, meaning that the formula should be created once and than dragged to the right or the bottom. This is illustrated in Figure 2.13. It displays the formulas in the R1C1 notation to show that all the formulas on a single row have the same structure. However, they are not identical. The formulas differ in their references to other cells. In this example a single formula was copied to 70 columns. If a spreadsheet is designed in accordance with this rule it means that to understand the sheet we do not have to inspect 71 cells to check if the formulas differ somewhere. We just have to inspect the cell in (in this case) the 10th column. In addition, because the column structure across the different worksheets is consistent this holds true for every worksheet. The guidelines of consistent formulas and defining a calculation only once, explain the dramatic decrease in the occurrence of the

Duplicate formula smell.



	9	10	11	12	13	14	15
29		0,6	0,6	0,6	0,6	0,7	0,7
30		= R[-2]C6 * R[-1]C	= R[-2]C6 * R[-1]C	= R[-2]C6 * R[-1]C	= R[-2]C6 * R[-1]C	= R[-2]C6 * R[-1]C	= R[-2]C6 * R[-1]C
31							
32							
33		0,7	0,7	0,7	0,7	0,7	0,7
34		= R[-2]C6 * R[-1]C	= R[-2]C6 * R[-1]C	= R[-2]C6 * R[-1]C	= R[-2]C6 * R[-1]C	= R[-2]C6 * R[-1]C	= R[-2]C6 * R[-1]C
35							

	I	J	K	L	M	N	O
29		0,6	0,6	0,6	0,6	0,7	0,7
30		= $F28 * J29	= $F28 * K29	= $F28 * L29	= $F28 * M29	= $F28 * N29	= $F28 * O29
31							
32							
33		0,7	0,7	0,7	0,7	0,7	0,7
34		= $F32 * J33	= $F32 * K33	= $F32 * L33	= $F32 * M33	= $F32 * N33	= $F32 * O33
35							

Figuur 2.13: Example of consistent formulas. The formulas in row 30 and 33 are displayed both in R1C1 (top) and A1 (bottom) notation. Although the formulas differ in normal notation, the R1C1 notation shows that they are actually identical.

The number of smell occurrences decreases for four of the five smells. However, the median for the *Long Calculation Chain* in the F1F9 dataset is slightly higher than in the Client dataset. This is not unexpected, trying to minimize the *Multiple References* and *Multiple Operations* smells (ie breaking long formulas in shorter parts), inevitable leads to longer calculation chains.

2.4.2 Size, Coupling, and Use of Functions

Based on the size metrics, we can conclude that the F1F9 models grew in size. Within the FAST standard there are several guidelines that could explain this increase.

- *Separate worksheets per functional class*: the standard dictates a strict separation between input (Foundation), calculation (Workings), output (Presentation), and control, which causes more worksheets per spreadsheet.

- *Maintain consistent column structure across all sheets*: Most financial models are time-related. For example, to calculate the business case for a major investment it is necessary to predict the future cash flows over the life span of the investment (which could easily be 30 years). The FAST standard prescribes to model the time-dimension along the columns of a worksheet. Because of this guideline, these time series are repeated on every worksheet even if that means that on some worksheets these columns are unused. Figure 2.11 shows an example of such a consistent column structure across different sheets.

- *Construct all calculations in a separate calculation block*: A calculation block (see Figure 2.12 for an example) consists of all the ingredients (inputs) that are necessary for a calculation. Ingredients can also be used in other calculations. The standard dictates that in such a case the ingredients are repeated (by direct linking) to form a new calculation block. This increases the number of non-empty cells on a worksheet.

With respect to coupling, the number of interworksheet connections was the only metric in the F1F9 sheets that differed from the Client sheets. This could be caused by the way calculation blocks are constructed according to the FAST standard. As we already saw, a calculation block consists of the necessary ingredients and the calculation itself. The ingredients are often input values that are coming from another (Foundation) worksheet and thus creating an interworksheet connection. If the same ingredient is necessary for another calculation, it will be repeated. However, the FAST standard forbids a series of linked links, so called daisy chains. We illustrate this concept with a small example in which for a certain calculation a start date is needed. The start date is coming from the sheet 'InpC' (which is a Foundation sheet) and is located in cell F11. The start date is an ingredient for a calculation block and it is put in cell F12 on the sheet 'Time'. The formula for this cell becomes:

```
F12: = InpC!F\$11
```

This same start date is also needed as ingredient in a second calculation block (in for example cell F21) on the same sheet. In general users tend to solve this with:

```
F21: = F12
```

However, this creates a daisy chain. The value from cell F11 on the sheet 'InpC' is retrieved via cell F12 on the sheet 'Time'. To prevent daisy chaining the formula should be:

```
F21: = InpC!F\$11
```

Applying this guideline will create an additional interworksheet connections every time an input value is re-used in a different calculation block. It explains why the number of interworksheet connections in the F1F9 sheets is higher than in the Client sheets.

The use of functions is similar in both datasets. In earlier work [39] we saw that in the Enron Corpus the majority of formulas only make a direct reference to a few other cells, are hardly nested and within the formula only one function (not being an operator) is used. Despite the fact that both the Client and the F1F9 dataset consists of complex financial models, we see the same kind of metrics with respect to the complexity of the formulas. Users tend to create simple formulas. Complex and large formulas do exists but are an exception.

2.5 Discussion

In the previous sections, we have analyzed the occurrences of smells and metrics with respect to size, coupling and use of functions in both the Client and the F1F9 spreadsheets. In this section, we discuss some topics that could affect the applicability and suitability of the approach used and the results found.

2.5.1 Threats to Validity

The dataset we received from F1F9 gave us a unique opportunity to work with complex, real-life, industrial spreadsheets to investigate what characterizes a smelly spreadsheet. Unfortunately, this real-life dataset comes with the price of reduced repeatability. We are strong believers of open data, but because the spreadsheets contain confidential information, we were only allowed to analyze them automatically and we are not able to share them.

A threat to the external validity of our analysis is the representativeness of the provided dataset. Half of the spreadsheets were created by a single company. However, the Client spreadsheets are from different clients. More important is the fact that all spreadsheets are financial models. Consequently the findings of our analysis can only be applied to spreadsheets within the specific domain of financial modeling.

2.5.2 Pivot Tables, Charts and VBA code

In our analysis, we limited ourselves to the described smells and metrics. However the improvements that were made by F1F9 could also affect the use of more elaborate structures like Pivot tables, charts and VBA code. In future research, we plan to specifically analyze these constructs.

2.5.3 Calculation Chains

All the smells that we have analyzed are calculated on the formula level. The same is true for the following metrics that we used to analyze the size, level of coupling and use of functions:

- s5 length of formula in characters
- c3 path depth per formula
- c4 total number of transitive precedents per formula
- f1 number of unique functions per formula
- f2 parse three depth per formula
- f3 number of preceding cells per formula

We took the single formula as the object of analysis. We considered it as the equivalent of a line of code. However, in a spreadsheet it is always possible to take a formula and split it over more than one cell. What we consider a line of code is actually an arbitrary decision of the user. To see and understand the complete code for a certain calculation, you need to look at the complete calculation chain. In future research, we plan to extend the analysis of smells and metrics to the level of the calculation chain.

2.6 Related Work

Our work builds upon the work of Hermans *et. al.* [27], in which the concept of spreadsheet smells at the formula level was introduced. However, for their evaluation they used a small dataset of which it was not known in advance whether it contained smelly spreadsheets. In addition to that paper, Hermans also worked on other types of spreadsheet smells, for example focusing on detecting smells between worksheets, rather than within [28]. Other work on spreadsheet smells was done by Cunha *et al.* who aim at detecting smells in values, such as typographical errors and values not following the normal distribution [38].

A second category of related work aims at defining spreadsheet metrics. Bregar developed a catalog of spreadsheet metrics based on software metrics [40]. He however did not evaluate his metrics in practice. Hodnigg and Mittermeir [41] also proposed several spreadsheet metrics of which some are similar to Bregar's. Their metrics are divided into three categories: general metrics, such as the number of formulas and the number of distinct formulas; formula complexity metrics, such as the number of references per formula, and the length of the longest calculation chain; and finally metrics, such as the presence of scripts in, e.g., Visual Basic for Applications (VBA), user defined functions and external sources. Hole *et al.*[42] propose an interesting approach to analyze spreadsheets in terms of basic spreadsheet metrics, such as the number of functions used, the presence of charts and the complexity of program code constructs with the specific aim of predicting the level of the spreadsheet creator.

A research direction related to smell detection is spreadsheet refactoring, which also has been addressed by researchers in recent years. The first to present a semi-automated approach were Badame and Dig [16], whose refactorings unfortunately were not directly based on smells. A generalization of this idea was presented by Hermans and Dig [15].

2.7 Concluding Remarks

This paper describes the analysis of a new spreadsheet dataset. This set consists of 54 pairs of spreadsheets, which both implement the same functionality, but are either smelly and hard to maintain (client) or well-structured (F1F9).

For each spreadsheet, we calculated the metrics that indicate formula smells and extended this analysis with additional metrics for size, coupling and the use of functions. For each metric we determined whether there was a significant difference between the client and the F1F9 sheets. If a difference was found, we calculated the effect size.

Based on this analysis we answered our two research questions:

R1 Do spreadsheets that are perceived as easier to understand, contain fewer smelly cells than spreadsheets that are perceived as problematic?

R2 What are the differences with respect to size, level of coupling and the use of functions between spreadsheets that are perceived as easier to understand and spreadsheets that are perceived as problematic?

Our analysis reveals two interesting points. Firstly, the F1F9 spreadsheets indeed suffer from smells to a much lower extent than the client sheets. We observed for example that the F1F9 sheets contain fewer duplicated formulas and that formulas have fewer references to other cells. Secondly, size and coupling metrics, obvious candidates to measure spreadsheet complexity, do not succeed in differentiating between the both parts of the datasets.

Our current analysis gives rise to ample directions for future work. In this paper we did a pairwise comparison on spreadsheet level. Because F1F9 rebuilt the models from scratch, it was not possible to do this on formula level. In future research, we are planning to analyze the effect of refactoring spreadsheet formulas in existing models. In such a case pairwise comparison on formula level is possible.

A spreadsheet that contains fewer smells should be easier to understand and maintain. In this study, we saw indeed that the spreadsheets improved by F1F9 contain fewer smells. However, it was not possible to interview the users of these spreadsheets to obtain their opinion about the understandability and maintainability of the spreadsheets. We believe this is a promising avenue for future research. We will perform a case study with users to test if they are able to understand and maintain a refactored spreadsheet with less effort.

3
The Effect of Delocalized Plans on Spreadsheet Comprehension: A Controlled Experiment

Spreadsheets are widely used in industry. Spreadsheets also suffer from typical software engineering issues. Previous research shows that they contain code smells, lack documentation and tests, and have a long live span during which they are transferred multiple times among users. These transfers highlight the importance of spreadsheet comprehension. Therefore, in this paper, we analyze the effect of the organization of formulas on spreadsheet comprehension.

To that end, we conduct a controlled experiment with 107 spreadsheet users, divided into two groups. One group receives a model where the formulas are organized such that all related components are grouped closely together, while the other group receives a model where the components are spread far and wide across the spreadsheet. All subjects perform the same set of comprehension tasks on their spreadsheet.

The results indicate that the way formulas are located relative to each other in a spreadsheet, influences the performance of the subjects in their ability to comprehend and adapt the spreadsheet. Especially for the comprehension tasks, the subjects perform better on the model where the formulas were grouped closely together. For the adaptation tasks, we found that the length of the calculation chain influences the performance of the subjects more than the location of the formulas itself.

This chapter has been published as 🗎 *B. Jansen and F. Hermans. The effect of delocalized plans on spreadsheet comprehension: a controlled experiment, ICPC17* [30]

The use of spreadsheets in industry is widespread. For millions of employees, spreadsheets form the day-to-day tool to solve business questions, create reports, and deliver support for planning and scheduling activities.

Spreadsheets can be considered a successful end-user programming language. It could also be argued that spreadsheets are more than an end-user programming language and that they *are* code [14], as there are many similarities with code. Like code, spreadsheets implement concepts like composition, selection and repetition. In spreadsheets, simple objects can be combined into more complex ones by including references to other cells within a cell's formula, they implement selection with an if-then-else structure and they mimic iterations by the replication of the same formula across many rows or columns.

However, like code, spreadsheets suffer from software maintenance issues. They lack documentation [37], contain code smells [27] and clones [43], have an average lifespan of five years and are used or maintained by an average of twelve different users [37].

There is a growing body of research in which methods of software engineering are transferred to spreadsheets. This research include testing [44], reverse engineering [45], [46], code smells, [27], [38], and refactoring [15], [16].

Because of their average lifetime of 5 years, spreadsheets are transferred multiple times from one user to another. These transfer scenarios stress the importance of spreadsheet comprehension. With respect to source code we know that the concept of delocalized plans or locality is a factor that influences program comprehension. There are several studies that indicate that delocalization negatively correlates with program comprehension [47], [48], [49], [50]. For spreadsheets this concept of locality is also relevant. The advice we can extract from several spreadsheet guidelines with respect to locality is conflicting. Conway and Ragsdale [51] stated in their spreadsheet guidelines that "things which are logically related should be arranged in close physical proximity and in the same columnar or row orientation". However, the FAST standard [36] prescribes to separate inputs from calculations and to put them on different worksheets. The first approach adheres to the concept of locality, the second approach will lead to delocalized plans. What is the effect of this design choice on the overall comprehensibility of the spreadsheet? It is this question that leads us to the core question of this research paper: **What is the effect of delocalized plans on spreadsheet comprehension?**

To address this goal, we set up a controlled experiment with 107 spreadsheet users. We create two different versions of a spreadsheet model, that differ in the organization of the formulas within the spreadsheet. In one model the formulas are closely grouped together while in the other model they are spread over multiple worksheets. The subjects are divided over the two models and asked to perform the same set of comprehension tasks. During the experiment, we measure 1) correctness, 2) perceived difficulty, 3) the time to completion, and 4) number of clicks needed for completion of the task.

The results reveal that the existence of delocalized plans in spreadsheets influences the user's ability to comprehend and adjust the spreadsheet. When users have to execute comprehension tasks on formulas with longer calculation chains they perform significantly better on the model that contained less delocalized plans.

The contributions of this paper are:

- A definition of the concept of delocalized plans in spreadsheets (Section 3.1)

- Design of a controlled experiment with 107 spreadsheet users to analyze the ability of subjects to comprehend and adjust a spreadsheet (Section 3.2)
- Translation of the software comprehension tasks as defined by Pacione *et al.* [52] to the spreadsheet domain (Section 3.2)
- An empirical evaluation of the effect of delocalized plans in spreadsheets on spreadsheet comprehension (Section 3.3)

We organize the remainder of this paper in the following way. In the next section, we provide background information about the concept of delocalized plans in the context of spreadsheets. In Section 3.2 we describe the set-up of the experiment followed by a presentation of the results in Section 3.3 and a discussion of the results in Section 3.4. The paper is concluded with an overview of the related work (Section 3.5) and the concluding remarks (Section 3.6).

3.1 Delocalized Plans in Spreadsheets

3.1.1 Delocalized Plans in Source Code

The concept of locality in source code is a well-researched area. Weinberg [47] defines the concepts of locality and linearity and their effect on program comprehension. *Locality* means that all relevant parts of the program are located closely together. *Linearity* means that decisions in the program are arranged in a strictly linear sequence. The two concepts are related: arranging decisions in a non-linear sequence often causes non-locality. Locality and linearity can support a programmer with the comprehension and adaption of code.

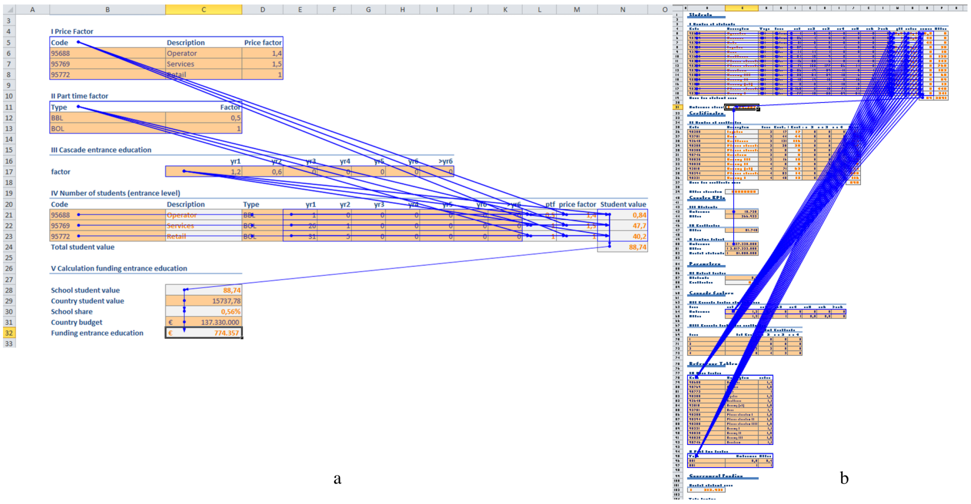
Letovsky and Soloway [48] reached a similar conclusion. They explore the relation between program comprehension and delocalized plans. A plan is defined as the technique that is used to realize the intention behind the code. A delocalized plan means that the code for the plan is spread far and wide in the source code. In their study, they concluded that delocalized plans are more liable to misinterpretations.

3.1.2 Translating Delocalized Plans to Spreadsheets

The concept of delocalized plans can easily be translated to spreadsheets. In the left part (a) of Figure 3.1 we have highlighted the formula in cell C32. It calculates the total funding a school receives for its entry level students. The formula in cell C32 is:

```
= C30 * C31
```

Cells C30 and C31 are the direct precedents of this formula and are located close to the formula itself (in the same column in the two rows directly above it). However, the calculation chain does not stop there. Both cells C30 and C31, in turn, also contain a formula and they again refer to other cells. To completely understand the calculation you need to trace back not only the direct but also the indirect precedents. They are illustrated in Figure 3.1 with the blue arrows. The indirect precedents are located somewhat further



I Price Factor

Code	Description	Price factor
95688	Operator	1,4
95769	Services	1,5
95772	Retail	1

II Part time factor

Type	Factor
BBL	0,5
BOL	1

III Cascade entrance education

	yr1	yr2	yr3	yr4	yr5	yr6	>yr6
factor	1,2	0,6	0	0	0	0	0

IV Number of students (entrance level)

Code	Description	Type	yr1	yr2	yr3	yr4	yr5	yr6	>yr6	ptf	price factor	Student value
95688	Operator	BBL	1	0	0	0	0	0	0	0,5	1,4	0,84
95769	Services	BOL	26	1	0	0	0	0	0	1	1,5	47,7
95772	Retail	BOL	31	5	0	0	0	0	0	1	1	40,2
Total student value												88,74

V Calculation funding entrance education

School student value		88,74
Country student value		15737,78
School share		0,56%
Country budget	€	137.330.000
Funding entrance education	€	774.357

Figuur 3.1: Two examples of the same formula. In the left example (a) all precedents are grouped closely together, the size of the model is 14 columns by 32 rows. In the example on the right (b), the precedents are spread far and wide in the model. The size of the model is 16 columns by 104 rows.

away. Nevertheless, they still can be presented in a readable manner on a single screen. The complete model spans 14 columns by 32 rows.

In the right part (b) of Figure 3.1 we illustrate the same calculation, with the same input and output, but in a spreadsheet where the formulas are organized in a different way. The calculation in this model is located in cell C21 and its formula is:

```
= O19 / C43 * C50
```

If we limit our attention to the direct precedents, we already can observe that they are located much farther apart than in the first example. They are located in different columns (O and C) and also the vertical distance is larger. The precedents are located two rows above, 22 rows below, and 29 rows below the formula. If we also include the indirect precedents, the situation deteriorates. It is going to be very difficult to present them all in a readable manner on a single screen. This version of the model spans 16 columns by 104 rows.

However, in a spreadsheet, there is a third dimension. In the previous two examples, the precedents of a formula were all located on the same worksheet. Yet, it is also possible that precedents are located on a different worksheet. Previous research [28] defines the Feature Envy code smell in the context of spreadsheets. A formula suffers from Feature Envy if it makes references to cells on different worksheets. The authors argue that refactoring such a formula by moving it to that different worksheet *"will likely improve understandability, since the formula is then closer to the cells it is referring to."*

Letovsky and Soloway use the term delocalized plan if the code for a plan is *"spread far and wide in the text of the program"*. If we apply this to spreadsheets, a delocalized plan would be a formula that has its precedents spread widely across the spreadsheet. This could mean that the precedents are located far apart on the same worksheet or that they are located on different worksheets. We regard a formula as delocalized if it is impossible to get an overview of all its precedents in a single glance.

Both Letovsky and Soloway and Weinberg argue that source code that tries to avoid delocalized plans is easier to comprehend and adapt. Is this also true for spreadsheets? To answer this question we have designed a controlled experiment. The set-up of this experiment is discussed in the next section.

3.2 Experimental Setup

The goal of this paper is to answer the question: Are spreadsheets that contain delocalized plans harder to understand? To address this goal we have formulated the following research questions. How does the existence of delocalized plans in spreadsheets influence the user's ability to:

RQ1 understand a component of a spreadsheet?

RQ2 understand the complete calculation model of a spreadsheet?

RQ3 adapt a spreadsheet?

The distinction between RQ1 and RQ2 is the level of abstraction. If we would explain the difference between RQ1 and RQ2 in the context of source code, then RQ1 is about code explanation and RQ2 about system understanding [53].

To answer these questions we design a controlled experiment. In the remainder of this section, we explain its set-up.

3.2.1 Subjects

To recruit subjects for the experiment we invite the participants of one of our MOOCs (Massive Open On-line Course), post it on social media and announce it via the mailing lists of EUSPRIG and our own research website. People who are interested are randomly assigned to one of the two models. In total, we received 107 valid responses.

We ask the participants to assess their own Excel skills and how frequently they use Excel on a five-point Likert scale. Table 3.1 gives an overview of how the participants are distributed over the two models and the average score for their Excel skills and the frequency in which they use Excel.

Tabel 3.1: Distribution of Participants, frequency and skill level were measured on a five point Likert Scale (1 = low, 5 = high)

	Model C	Model F
# Subjects	56	51
Frequency	3.3	3.4
Skill Level	3.6	3.6

3.2.2 Tasks

We ask each subject to answer nine questions about the spreadsheet. The questions are the same in both models. In order to answer these questions the participants have to analyze the formulas in the spreadsheet, make small changes and correct errors. To obtain a proper set of comprehension tasks we use the framework that was defined by Pacione *et al.* [52], commonly used for empirical evaluation of code comprehension [54],[53]. They analyzed sets of software comprehension tasks that were suggested in software visualization and comprehension evaluation literature and classified them in nine distinct software comprehension activities. Table 3.2 gives an overview of these nine categories.

Tabel 3.2: Comprehension activities from Pacione *et al.* [52]

Activity	Description
A1	Investigating the functionality of (a part of) the system
A2	Adding to or changing the system's functionality
A3	Investigating the internal structure of an artefact
A4	Investigating dependencies between artefacts
A5	Investigating runtime interactions in the system
A6	Investigating how much an artefact is used
A7	Investigating patterns in the system's execution
A8	Assessing the quality of the system's design
A9	Understanding the domain of the system

Our questions[1] are designed in such a way that each of them addresses, at least, one of the categories and that all, for spreadsheet relevant categories, are covered. Table 3.3 shows for each question in the experiment one or more of Pacione's activities that are related to the question in the experiment.

Tabel 3.3: Comprehension tasks used in experiment and their mapping to both the nine comprehension activities of Pacione and the research questions of this paper

Question	Task	RQ	A1	A2	A3	A4	A5	A6	A7	A8	A9	Total
Q1	Explain a calculation	RQ2	X		X						X	3
Q2	Find and correct an error	RQ3		X	X							2
Q3	Adapt a calculation	RQ3	X	X	X							3
Q4	Explain a key concept of the model	RQ2	X		X						X	3
Q5	Determine relationship between two cells	RQ1				X		X				2
Q6	Find dependents of a cell	RQ1				X		X				2
Q7	Explain how the spreadsheet can be improved	RQ2			X					X		2
Q8	Assess adaptability of the spreadsheet	RQ3								X		1
Q9	Assess transferability of the spreadsheet	RQ2								X		1
Total			3	2	5	2	-	2	-	3	2	

Table 3.3 shows that there are no tasks in the experiment that are related to A5 (Investigating runtime interactions in the system) and A7 (Investigating patterns in the system's execution). Because of the directness of a spreadsheet, there is no clear distinction between coding and runtime. Therefore, we exclude these two activities from the experiment.

3.2.3 The Spreadsheet model

To ensure that all the different comprehension tasks are covered in the experiment we need a spreadsheet of a reasonable size with non-trivial calculations. Furthermore, the

[1]Both the questions and the spreadsheet models used in this experiment can be found at: `https://doi.org/10.6084/m9.figshare.3167983`

experiment should simulate a realistic scenario. For these reasons we choose a spreadsheet that is used in practice by schools to calculate the total funding they receive from the Dutch government, based on the number of students and the number of certificates they issue.

We created two versions of the model, one that adheres to the concept of locality (Model **C**, precedents are located **C**lose to the formula) and one that contains numerous of delocalized plans (Model **F**, precedents are located **F**ar from the formula). The delocalized plans were created by grouping related data on separate worksheets, a practice that is often used by spreadsheet users to structure data in a spreadsheet. We tried to keep the formulas in both models as much the same as possible. However, the layout of the data in a spreadsheet and the structure of the formulas are highly intertwined. Because of this we had to change some of the formulas. Figure 3.2 is a screen shot of Model C. The blue arrows visualize the precedents that are used in the formula of the active cell. They are all located on the same worksheet in the rows above the formula.

In Figure 3.3 the same formula is displayed, but now in Model F. The precedents are not located on the same worksheet but spread over four different worksheets.

To inspect the formula the user has to switch back and forth between four different worksheets. This is in contrast to the first example (Figure 3.2, Model C), in which all precedents are on the same worksheet and because they are located close to the formula itself it is not even necessary to scroll when inspecting the formulas.

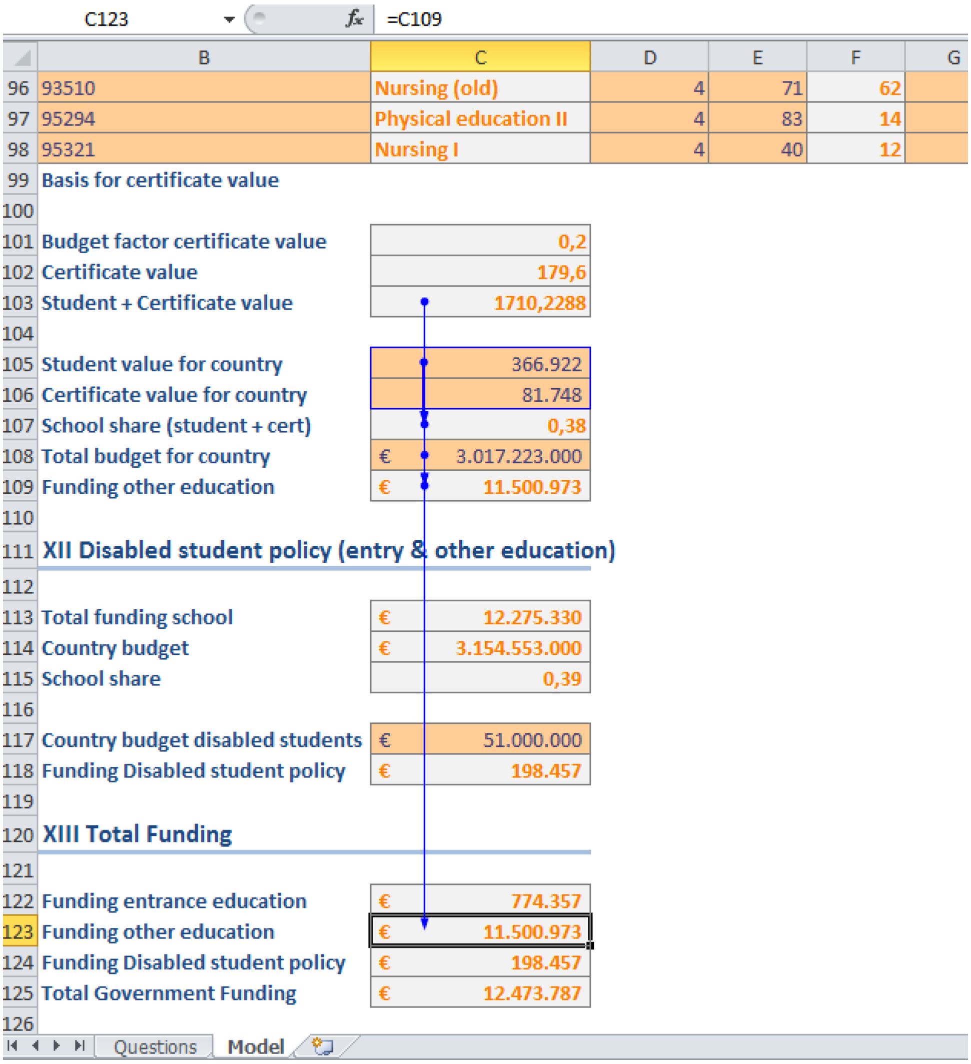
In a spreadsheet with delocalized plans formulas make reference to cells that are located far from the formula, often on a different worksheet. The number of external references (to another worksheet) is also used to determine if a spreadsheet is suffering from the Feature Envy smell. To be sure that the two spreadsheets differ sufficiently from each other we analyzed both models on the occurrence of the Feature Envy smell.

As a metric for enviousness we counted all references a formula has to cells contained by the other worksheet which is the same definition that was used in [28]. For both spreadsheets we divided the number of these external references by the total number of references for all unique formulas. Model C scored 0%, Model F 73%.

To determine if one of the models is suffering from Feature Envy we have to establish a threshold for the described metric. We established this threshold by analyzing the distribution of the metric over the Enron Spreadsheet Corpus [10] and used the approach of Alves *et al.* [55]. In total we analyzed 9.080 spreadsheets with a total of 770.644 unique formulas. 90% of the analyzed spreadsheets had a score for Feature Envy of 16% or less. In accordance with [55] we defined this score of 16% for the metric as a very high risk that the spreadsheet is suffering from the Feature Envy smell. Based on this threshold Model F is suffering from Feature Envy and Model C not.

3.2.4 Experimental procedure

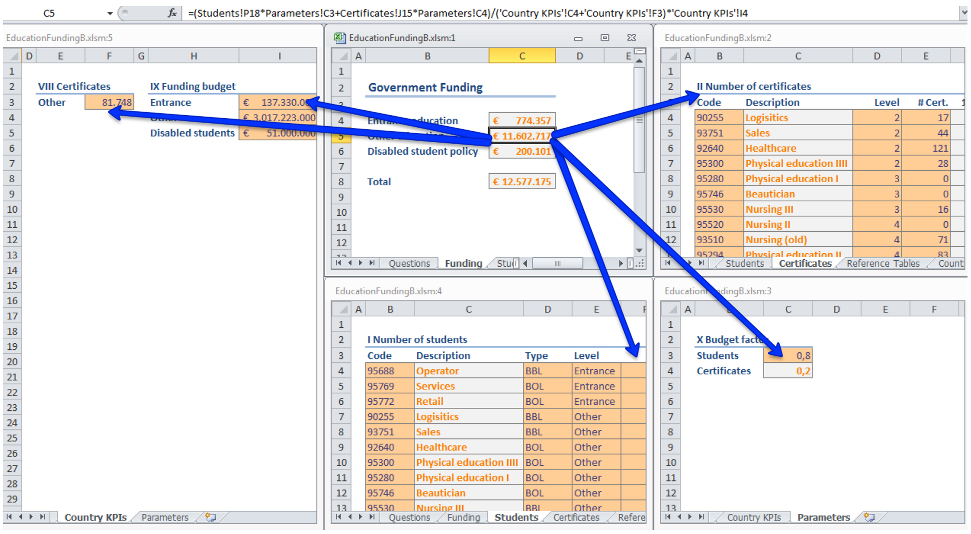
To participate in the experiment the subjects have to register themselves on the web page of the experiment and after registration, they are randomly redirected to a download link for either Model C or Model F. In the instructions we ask them to answer all questions in one sitting. We added a VBA script to the spreadsheet that logs all activities of the subjects. The script is activated when a participant clicks on a random cell in the spreadsheet. From that moment on, each click, activation of a worksheet, or change to a cell is logged with a timestamp. As soon as a subject closes the workbook the log file is automatically sent to



C123 =C109

	B	C		D	E	F	G
96	93510	Nursing (old)		4	71	62	
97	95294	Physical education II		4	83	14	
98	95321	Nursing I		4	40	12	
99	Basis for certificate value						
100							
101	Budget factor certificate value		0,2				
102	Certificate value		179,6				
103	Student + Certificate value		1710,2288				
104							
105	Student value for country		366.922				
106	Certificate value for country		81.748				
107	School share (student + cert)		0,38				
108	Total budget for country	€	3.017.223.000				
109	Funding other education	€	11.500.973				
110							
111	XII Disabled student policy (entry & other education)						
112							
113	Total funding school	€	12.275.330				
114	Country budget	€	3.154.553.000				
115	School share		0,39				
116							
117	Country budget disabled students	€	51.000.000				
118	Funding Disabled student policy	€	198.457				
119							
120	XIII Total Funding						
121							
122	Funding entrance education	€	774.357				
123	Funding other education	€	11.500.973				
124	Funding Disabled student policy	€	198.457				
125	Total Government Funding	€	12.473.787				
126							

Figuur 3.2: Example of a spreadsheet designed with concept of locality in mind



C5 =(Students!P18*Parameters!C3+Certificates!J15*Parameters!C4)/('Country KPIs'!C4+'Country KPIs'!F3)*'Country KPIs'!I4

VIII Certificates		IX Funding budget	
Other	81.748	Entrance	€ 137.330.00
			€ 3.017.223.000
		Disabled students	€ 51.000.000

Government Funding

Entrance education	€ 774.357
	€ 11.602.717
Disabled student policy	€ 200.101
Total	€ 12.577.175

II Number of certificates

Code	Description	Level	# Cert.
90255	Logisitics	2	17
93751	Sales	2	44
92640	Healthcare	2	121
95300	Physical education IIII	2	28
95280	Physical education I	3	0
95746	Beautician	3	0
95530	Nursing III	3	16
95520	Nursing II	4	0
93510	Nursing (old)	4	71
95294	Physical education II	4	83

I Number of students

Code	Description	Type	Level
95688	Operator	BBL	Entrance
95769	Services	BOL	Entrance
95772	Retail	BOL	Entrance
90255	Logisitics	BBL	Other
93751	Sales	BBL	Other
92640	Healthcare	BOL	Other
95300	Physical education IIII	BOL	Other
95280	Physical education I	BOL	Other
95746	Beautician	BOL	Other
95530	Nursing III	BBL	Other

X Budget factor

Students	0,8
Certificates	0,2

Figuur 3.3: Example of spreadsheet containing delocalized plans

us. In this set-up, we cannot control the subject's behavior. It is plausible that the subject will be interrupted during the experiment and is not able to complete all the tasks in one sitting. However, using the timestamps we could deduce if there was a continuous time line. From the results, we learned that in 99% of the cases the time between two activities (a click or a change in a cell) is not more than one and a half minute. We, therefore, ignored all gaps between activities that lasted more than one and half minute, to ensure a proper timing of the tasks.

3.2.5 Variables and Analysis Procedure

The independent variable in this experiment is the existence of delocalized plans in a spreadsheet. There are several dependent variables. The first is the correctness of the answer. We measure this by checking all answers against our answer model and grading it on a scale of one to ten (where ten is a perfect score).

The second dependent variable is the perceived difficulty of the question. After each question, we ask each subject to indicate on a five-point Likert scale how difficult they thought the question was.

The other dependent variables in the experiment are the time, the number of clicks, and the number of worksheet changes a subject needs to answer a question. We choose to measure the number of clicks because the most common way to analyze a formula in a spreadsheet is to select the cell with the formula, which then can be inspected in the formula bar. Every cell selection leads to a click and gives us information about the formulas the user analyzed during a specific task. The VBA script measures the aforementioned variables. When a subject closes the workbook the associated log file is automatically sent to us by e-mail.

In the next section, we present the results of our experiment and answer our research questions.

3.3 Results

As described in the previous section, we analyzed five different variables: correctness, perceived difficulty, time, clicks, and worksheet changes. During the experiment the subjects had to answer nine different questions. Table 3.4 gives an overview per question of the results for these variables.

Tabel 3.4: Average values for dependent variables correctness, difficulty, time, # clicks, # sheet changes for both Model C and Model F

		Correctness†		Difficulty		Time		# Clicks		# Sheet Changes	
Question	Task	C	F	C	F	C	F	C	F	C	F
Q1	Explain a calculation	6.2	5.9	3.8	3.2	5:26	7:55	25	33	0	6
Q2	Find and correct an error	7.9	8.5	3.7	3.6	5:27	5:21	75	69	0	2
Q3	Adapt a calculation	5.4	6.8	3.5	3.6	6:44	5:05	65	19	0	4
Q4	Explain a key concept of the model	5.6	4.1	3.5	3.2	4:23	4:02	30	23	0	3
Q5	Determine relationship between two cells	8.6	8.9	3.4	3.4	6:42	5:57	91	58	0	6
Q6	Find dependents of a cell	6.8	5.9	3.5	3.6	3:51	4:28	39	87	0	7
Q7	Explain how the spreadsheet can be improved	-	-	3.1	3.3	2:28	1:50	25	5	0	3
Q8	Assess adaptability of the spreadsheet	6.5	6.2	-	-	1:14	1:04	2	1	0	0
Q9	Assess transferability of the spreadsheet†	5.3	5.2	-	-	0:50	0:53	1	0	0	0

† The scores for Q8 and Q9 do not represent the correctness of the answer but the assessment of the adaptability and transferability of the spreadsheet on a scale from one to ten, one meaning very difficult, ten very easy.

†† For the highlighted cells there is a significant difference between the two models for the combination of the question and the dependent variable.

The results are displayed separately for both spreadsheet models. Model C was designed in adherence to the concept of locality, trying to minimize the number of delocalized plans. This is in contrast to Model F where the precedents of formulas were spread far and wide in the spreadsheet, leading to a high number of delocalized plans (see for examples Figure 3.2 and 3.3 in Section 3.2).

A Shapiro-Wilk test showed that the data did not follow a normal distribution for almost all combinations of questions and models. Therefore, we used the Wilcoxon-Mann-Whitney test to determine per question if there was a significant difference between the two models. Table 3.5 shows that for some questions there is a significant difference between the two models.

For each variable with a significant difference we calculated the effect with the Cliff's Delta (Table 3.6). For most variables the measured effect was medium ($0.33 \le d < 0.47$) [56]. Exceptions are the number of clicks for Question 6 where the effect is large ($d \ge 0.47$) and the correctness, where the effect for Q3, Q4, and Q6 is small ($0.15 \le d < 0.33$).

In the remainder of this section, we will interpret the results and answer our research questions.

Tabel 3.5: p-values (Wilcoxon-Mann-Whitney) per question

Question	Correctness	Difficulty	Time	# Clicks
Q1	-	<0.05	<0.05	<0.05
Q2	-	-	-	-
Q3	<0.05	-	-	<0.05
Q4	<0.05	-	-	-
Q5	-	-	-	-
Q6	<0.10	-	<0.05	<0.05
Q7	-	-	-	-
Q8	-	-	-	-
Q9	-	-	-	-

Tabel 3.6: Cliff's Delta per question

Question	Correctness	Difficulty	Time	# Clicks
Q1	-	0.4044	0.4194	0.3000
Q3	0.2108	-	-	0.3955
Q4	0.2186	-	-	-
Q6	0.1835	-	0.3865	0.5100

3.3.1 RQ1: How does the existence of delocalized plans in spreadsheets influence the user's ability to understand a component of the spreadsheet?

Pacione considers several levels of abstraction with respect to code comprehension. We see this reflected in the set of comprehension activities (Table 3.2). In our experiment, the questions Q5 and Q6 are related to code comprehension on the component level (see Table 3.3).

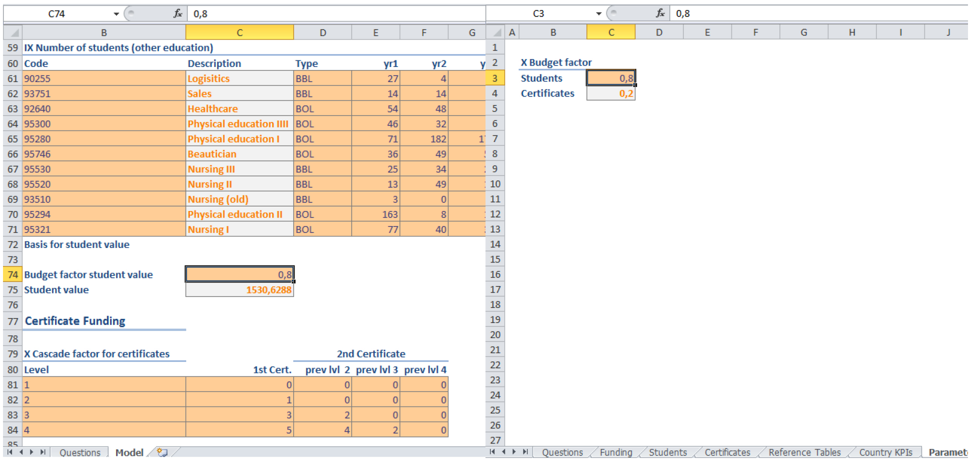
In Question 5 we ask the subjects to determine if one cell is a precedent of another cell. An example of such a question is:

> '*Answer the following statement with yes or no. The student value for entrance education (result of table IV, cell N24) is used in the calculation of the School student value (cell C28).*'

To answer this question the subjects have to inspect the formula in cell C28 to see if it makes a reference to cell N24. For this question, there is no significant difference in performance between the two groups. This in contrast to Question 6 where there is a significant difference. In this case the subjects have to answer the following question (see also Figure 3.4):

> '*Which cells are affected if you change the 'budget factor student value' (C74).*'

This question is different from Question 5. In Question 5 we provide two components and ask the subjects if they are related. In this question, we provide one component and ask the subjects to identify all components that are directly or indirectly related to this one. To be able to answer this question they have to trace all dependents of this cell. The participants in Model C answered this question significantly faster and needed fewer clicks. Notable is the high number of clicks that participants need to answer this question in Model F. In addition of these clicks they also needed to change frequently between worksheets.



C74 | 0,8

	B	C	D	E	F
59	IX Number of students (other education)				
60	Code	Description	Type	yr1	yr2
61	90255	Logisitics	BBL	27	4
62	93751	Sales	BBL	14	14
63	92640	Healthcare	BOL	54	48
64	95300	Physical education IIII	BOL	46	32
65	95280	Physical education I	BOL	71	182
66	95746	Beautician	BOL	36	49
67	95530	Nursing III	BBL	25	34
68	95520	Nursing II	BBL	13	49
69	93510	Nursing (old)	BBL	3	0
70	95294	Physical education II	BOL	163	8
71	95321	Nursing I	BOL	77	40
72	Basis for student value				
73					
74	Budget factor student value	0,8			
75	Student value	1530,6288			
76					
77	Certificate Funding				
78					
79	X Cascade factor for certificates		2nd Certificate		
80	Level	1st Cert.	prev lvl 2	prev lvl 3	prev lvl 4
81	1	0	0	0	0
82	2	1	0	0	0
83	3	3	2	0	0
84	4	5	4	2	0

Questions | Model

C3 | 0,8

	B	C
1		
2	X Budget factor	
3	Students	0,8
4	Certificates	0,2

Questions | Funding | Students | Certificates | Reference Tables | Country KPIs | Paramet

Figuur 3.4: Budget factor student value in both Model C (left) and Model F (right)

Based on these results we observe that the existence of delocalized plans in spreadsheets does influence the user's ability to comprehend the relations between components of the spreadsheet. If there are more delocalized plans, the user needs more time and is less accurate. However, we only see this effect when users are inspecting longer calculation chains.

3.3.2 RQ2: How does the existence of delocalized plans in spreadsheets influence the user's ability to understand the complete calculation model of the spreadsheet?

Question 1 and Question 4 test to which degree the subject understands the workings of the model. As stated in the previous section, the model calculates the total funding a school receives given the number of students. The main component of the funding is the student value. It is depending on the number of students but has a different value. In Question 1 we ask the subject to explain why the student value is higher than the total number of students. To answer this question they have to analyze a set of formulas and the relations between them.

For both models, the correctness of the answer did not show any difference. However, on average it took the participants in Model F more time and they needed more clicks than the participants in Model C. Also the Model F participants thought that the question was more difficult. In Model C all cells used to calculate the student value could be inspected on a single worksheet without the need for scrolling. In Model F the participants had to switch between two worksheets. It is plausible that the switching between sheets and the fact that all precedents of the formula could not be inspected on a single screen is responsible for the more time that was needed, the higher number of clicks, and a higher level of perceived difficulty.

In Question 4 we also asked the participants to explain the working of the model. An important concept in the model is the cascade factor. It is a weighting factor that ensures that the school gets less funding for students that need more time to finish their education. We ask the participants to explain the cascade factor in their own words. Although there is no significant difference in the time and number of clicks needed to answer this question, the quality of the answers in model C is significantly higher (See Table 3.5).

From these results, we conclude that the existence of delocalized plans in a spreadsheet influences the user's ability to comprehend the complete calculation model of the spreadsheet. However, we also observe that although we see a difference for both questions, the results differ per question. In Question 4 there is a difference in the correctness of the answers and in Question 1 it is the timing, perceived difficulty and the number of clicks. In future research, we will design a follow-up experiment to gain better insight into the causes for these differences.

In both Question 7 and 9, the subjects are asked to assess the quality of the spreadsheet. In Question 7 they have to suggest improvements to the model, whereas in Question 9 they are asked to assess how easy it is to explain the working of the spreadsheet to somebody else. Both questions are related to the user's ability to understand the spreadsheet.

It is notable that there is no significant difference, for both questions, in the subject's assessment of the quality of the spreadsheet, whereas we do observe a difference in the performance when subjects have to execute a comprehension task in Q1 and Q4.

We also received contradicting advice about how to improve the spreadsheet. A subject working with Model C (that consisted of a single worksheet) suggested to: "*split the long sheet to several sheets and name them by what they calculate.*", while a subject working with Model F (consisting of multiple worksheets) wrote that: "*having all the inputs on one sheet would be helpful (to prevent having to scroll trough tabs all the time)*".

In both groups, there were several subjects that suggested that the spreadsheet would be easier to comprehend if named ranges were used (replacing cell addresses by meaningful names). In future work, we will research the effect of using named ranges on formula comprehension.

3.3.3 RQ3 How does the existence of delocalized plans in spreadsheets influence the user's ability to adapt the spreadsheet?

In both Question 2 and 3, the subjects are asked to make adaptations to the model. In Question 2 they have to find and correct an error. For this question, we do not see any significant difference between the two groups. The subjects knew beforehand which part of the spreadsheet contained the error. To find the error it was not necessary to trace all the precedents, inspecting the formula was sufficient. This explains why the organization of the formulas in the spreadsheet did not influence the subjects ability to correct the error.

In Question 3 we asked the participants to change the model. Each school receives additional funding for disabled students. In our model, it takes into account both the number of students at entry level as well as the other students. In the question, we asked to exclude the entry level students from the calculation. In this case, the results indicate that it was easier to make this change in Model F than in Model C. For model F, the participants needed fewer clicks and submitted a higher number of correct answers. In Model C all components of the formula were located on a single sheet. In Model F the components were split between two worksheets. In order to make the change, the subjects needed to change sheets several times. Nevertheless, the performance in Model F was better.

There are two possible explanations. Although in Model F, the components of the formula were spread over two different sheets, on both sheets it was possible for the subjects to see all information in a single glance. Whereas in Model C all information was located on one sheet, but not visible in a single glance. We suspect that subjects had to scroll to see the complete picture. So one explanation could be that the ability to see all information in a single glance is responsible for the better performance.

Another explanation could be the formula itself. In order to make the change in Model C, the subjects needed to make changes in more than one cell, whereas in Model F it was sufficient to make a change in only one cell.

We designed a small second experiment to determine which of these two explanations is more plausible. We provided fourteen employees of the financial staff of Delft University of Technology with an advanced Excel training. During this training we let them do an exercise that was designed to determine which of the aforementioned explanations is the most plausible.

In this experiment, for both models, it was not possible to see all information in one glance and scrolling was necessary. The only remaining difference was the way the formula was constructed. The fourteen employees were randomly assigned to either Model C or Model F. The results of this experiment are summarized in Table 3.7.

Tabel 3.7: Results for spreadsheet experiment with TU Delft controllers

Activity	Model C	Model F
Correctness	8.1	10
Difficulty	3.2	3.4
Time	8:15	4:40
Clicks	40	9

Again the performance was better in Model F. All eight subjects that were assigned to Model F completed the exercise flawlessly. In this model it was sufficient to change just a single cell, making the way the formula was constructed the most plausible explanation for the better performance.

This finding also sheds light on a previously unresolved issue regarding the trade-off between the long method and the long calculation chain smell in spreadsheet formulas as discussed in [27] and [29]. Defining a short formula to avoid the long method smell inevitably leads to a longer calculation chain. Trying to reduce the long calculation chain smell will lead to longer formulas. Based on the findings in this paper it appears that reducing the length of the calculation chain has more impact on maintainability of a formula than the length of the formula itself. This contradicts the FAST [36] guideline to write short formulas. A short formula is maybe easier to understand, but it leads to longer calculation chains and the experiment shows that users make more errors when they have to make adjustments to the logic in the calculation chain. In future research, we will conduct an experiment with a larger test group to further validate these conclusions.

The last question that is related to the user's ability to adapt the model is Question 8. In this question subjects are asked to assess how easy it would be to adapt the spreadsheet. The results for this question do not show a significant difference between the two models.

The results do not provide evidence that the existence of delocalized plans influences the user's ability to adapt the spreadsheet. However, they do indicate that regardless the existence of delocalized plans, the length of a calculation chain does influence the user while adapting the spreadsheet. If there is a longer calculation chain, the user needs more clicks to make the change in the model and is more likely to make errors.

3.4 Discussion

3.4.1 Threats to validity

There are several threats to the internal validity of this study. The first threat is the subject's skill level with respect to Excel. To mitigate this risk we asked the subjects to rate how frequently they use Excel and how they assess their skill level on a five-point Likert scale. Table 3.1 shows that for both variables there is no significant difference between the two groups.

The second threat is that the subjects were aware of the goals of the Study. To mitigate this we referred to the goals of the study in very general terms in both the invitation to participate and the instruction (for example: *'We are very much interested in how people are interacting with spreadsheets'*). Furthermore, the log file we used to analyze the subject's interaction with the spreadsheet was sent to us without showing it to the subject.

A third threat to the internal validity is the threat of self-selection. We invited participants of a MOOC about spreadsheets, approached mailing list members of a spreadsheet interest group (EUSPRIG), and members of our own mailing list. Because of this, it is plausible that all subjects have a more than average interest in the subject of spreadsheets and are not representative of the total population of spreadsheet users. Nevertheless, we decided to approach these possible subjects to maximize the set of participants. And although these subjects have possibly a more than average interest in spreadsheets it is also more likely that they use spreadsheets in their daily activities.

A final threat to the internal validity could be a learning effect. We mitigated this by allowing subjects to participate only once in the experiment. If we received multiple log files from the same e-mail address, only the first submission was accepted for the experiment. Nevertheless, a learning effect also occurs because the subject acquires more knowledge of the model while answering the questions. We decided not to randomize the sequence of questions to mitigate this effect. Pacione [52] considers several levels of abstraction with respect to code comprehension and the sequence of our questions has been structured accordingly.

A threat to the external validity of the experiment could be the representativeness of the spreadsheet model that we used in the study. We mitigated this risk by using a spreadsheet model that is based on a model that is used in practice by professionals.

Also, the representativeness of the subjects is an external threat. This risk is reduced by the relatively large sample size of 107 subjects. They varied in their age, cultural background, and Excel skill level and were randomly assigned to one of the two models.

3.4.2 Effect of long calculation chains

When we started this study, we expected to find a clear relation between the existence of delocalized plans in spreadsheets and the user's ability to comprehend the spreadsheet. We did find such a relation, but it is more nuanced than we thought. It appears that also the length of a calculation chain is an important factor that influences spreadsheet comprehension. Furthermore, the type of comprehension task seems to be relevant. Subjects perform significantly better in spreadsheets without delocalized plans when they have to explain the working of the model. However, when we ask them to adapt the model, it is the length of the calculation chain that determines their performance and not the existence of delocalized plans.

3.4.3 Long Calculation Chain smell versus Multiple Operations and Multiple References smells

In previous research on code smells in spreadsheet formulas it is stated that there is a trade-off between the *Long Calculation Chain* smell and the *Multiple Operations* and *Multiple References* smells [27], [29]. Our results indicate that the *Long Calculation Chain* smell has a higher impact on the user's ability to comprehend and maintain a formula than the *Multiple Operations* and *Multiple References* smells. It is easier for a user to comprehend and adapt a long formula with a short calculation chain than a short formula with a long calculation chain. This finding contradicts the popular belief that short formulas are easier to comprehend. For example, the FAST Standard Organization advises in their popular FAST standard to write short formulas: *"A formula longer than your thumb likely means that it should be broken into more than one step."* [36].

3.5 Related Work

3.5.1 Software Engineering Methods and Spreadsheets

There are numerous studies that focus on applying software engineering methods on spreadsheets. Rothermel *et al.* brought the concept of testing to spreadsheets with the What You See is What You Test paradigm [44]. Other researchers focused on the domain of reverse engineering and came up with methods to extract class diagrams from spreadsheets [45] [46] or to visualize the data flow within spreadsheets [6]. Fowler introduced the concept of smells in code [35], but smells also exist in spreadsheets. They are described in detail in the work of Hermans [57] [28], Cunha [38] and Barowy *et al.* [58]. From smells, it is a small and logical step to refactoring. Hermans defined refactorings for formula smells in spreadsheets [15]. Inspired by this work Badame and Dig developed RefBook, a tool that supports a number of refactorings for spreadsheet formulas [16].

3.5.2 Delocalized Plans and Program Comprehension

Weinberg defines several principles for programming language design [47]. According to the author, a properly designed language aids the programmer by keeping relevant information close at hand. In this context, he defines the concepts of locality and linearity. Locality means that all relevant parts of a program are found in the same place, for example on the same page or on a single screen. Linearity means that the decisions in the program are arranged in a strictly linear sequence.

Letovsky and Soloway [48] studied the relation between locality and program comprehension. They state that the goal of program understanding is to recover the intentions behind the code. They define a plan as the technique that is used to realize an intention. In their paper they focus on so-called delocalized plans, meaning that the code for the plan is not closely grouped, but spread far and wide in the source code. They found that in order to understand a program, programmers make reasonable but sometimes incorrect assumptions. They tend to leave these assumptions unverified if the effort required to verify the assumption is great. Typically in delocalized plans, it will take more effort to verify the assumptions. According to the authors, these plans are more liable to misinterpretation than plans whose code is closely grouped. Delocalized plans could also be defined as plans with data flow links spanning widely separate parts of the code. Providing the developer with a data flow analyzer that makes these links explicit could reduce the risk of misinterpretations.

Constantine [50] took a different approach. He focused on the concept of localization in relation to user interface design. He defines the metric Visual Coherence that measures how the arrangement of visual components matches with their semantic relationships. He found that professional developers preferred more visually coherent designs and thought they were easier to use.

3.5.3 Controlled Experiments in Software Engineering

Other research discusses the set-up of experiments to analyze software comprehension. Pacione *et al.* [52] introduce a software visualization model for supporting object-oriented software comprehension. They evaluate the performance of visualization tools by assessing their performance in typical software comprehension tasks. Based on previous work and evaluation tasks found in software comprehension and software visualization literature, they introduced a framework of nine principal comprehension activities (see also Table 3.2 in Section 3.2). In the current paper, we use this framework to make sure that all relevant program comprehension activities were covered in our experimental setup.

A similar set-up of a controlled experiment in software engineering was used by Cornelissen *et. al* [54]. The authors use a controlled experiment to evaluate the effectiveness of a tool for the visualization of large traces and also apply Pacione's comprehension framework to select their comprehension tasks for the experiment.

3.6 Concluding Remarks

The goal of this paper is to answer the question: Are spreadsheets that contain delocalized plans harder to understand?

To address this goal, we set up a controlled experiment using two spreadsheets that differ in the organization of the formulas within the spreadsheet. In the experiment, the subjects are divided over the two models and perform a number of comprehension tasks, of which we measure correctness, perceived difficulty, completion time and the total number of clicks.

The results reveal that the existence of delocalized plans in spreadsheets influences the user's ability to comprehend and adjust the spreadsheet: subjects perform significantly better on the model that contained less delocalized plans when they have to perform comprehension tasks on formulas with longer calculation chains.

The contributions of this paper are:

- A definition of the concept of delocalized plans in spreadsheets (Section 3.1)
- Design of a controlled experiment to analyze the ability of subjects to comprehend and adjust a spreadsheet (Section 3.2)
- Translation of the software comprehension tasks as defined by Pacione *et al.* [52] to the spreadsheet domain (Section 3.2)
- An empirical evaluation of the effect of delocalized plans in spreadsheets on spreadsheet comprehension (Section 3.3)

3.6.1 Future work

This paper triggers several ideas for follow-up research. Firstly, a more extensive experiment is needed to understand the impact of delocalized plans in spreadsheets in more depth. For example, with a larger controlled experiment using multiple spreadsheets, followed by a longitudinal study.

Furthermore, we envision a tool that measures the occurrence of delocalized plans in spreadsheets. Such a tool could support users in estimating the effort needed for maintaining a spreadsheet.

In this experiment, references in formulas were presented in 'A1' style. However, there are alternatives like 'R1C1' notation or the use of named ranges for references. In a future study, we will investigate the effect of these different forms of notation on formula comprehension.

4 Detecting and Predicting Evolution in Spreadsheets

The use of spreadsheets in industry is widespread and the information that they provide is often used for decisions. Research has shown that spreadsheets are error-prone, leading to the risk that decisions are made on incorrect information.

Software Evolution is a well-researched topic and the results have proven to support developers in creating better software. Could this also be applied to spreadsheets? Unfortunately, the research on spreadsheet evolution is still limited. Therefore, the aim of this paper is to obtain a better understanding of how spreadsheets evolve over time and if the results of such a study provide similar benefits for spreadsheets as it does for source code.

In this study, we cooperated with Alliander, a large energy network company in the Netherlands. We conducted two case studies on two different set of spreadsheets that both were already maintained for a period of three years. To have a better understanding of the spreadsheets itself and the context in which they evolved, we also interviewed the creators of the spreadsheets.

We focus on the changes that are made over time in the formulas. Changes in these formulas change the behavior of the spreadsheet and could possibly introduce errors. To effectively analyze these changes we developed an algorithm that is able to detect and visualize these changes.

Results indicate that studying the evolution of a spreadsheet helps to identify areas in the spreadsheet that are error-prone, likely to change or that could benefit from refactoring. Furthermore, by analyzing the frequency in which formulas are changed from version to version, it is possible to predict which formulas need to be changed when a new version of the spreadsheet is created.

This chapter has been published as *B. Jansen, F. Hermans, and E. Tazelaar. Detecting and Predicting Evolution in Spreadsheets, ICSME18* [31]

The use of spreadsheets is widespread in industry. Panko [1] estimates that 95% of U.S. firms use spreadsheets in some form of financial reporting and Winston [2] estimates that 90% of all analysts in industry use spreadsheets for their calculations. Hermans *et al.* make compelling arguments that spreadsheets are code [14] and with an estimate of 500 million active users worldwide, spreadsheets are a successful end-user programming language. Notwithstanding their common use, research has also proven that spreadsheets are error-prone [7] which can lead to incorrect decisions and loss of money[1].

Until now, much of the spreadsheet research was focused on improving spreadsheets by applying software engineering methods to them. The concept of testing was brought to spreadsheets by Rothermel *et al.* [44] and recently Roy [20] investigated how users test spreadsheets. Hermans [45] and Cunha *et al.* [46] covered the topic of reverse engineering and proposed methods for extracting class diagrams from spreadsheets. Several studies were published about the existence of code smells in spreadsheets [57] [38] [58]. From code smells it is a small step to refactoring. Both Hermans and Dig [15] and Badame and Dig [16] proposed tools that support several refactorings for spreadsheet formulas.

Contrary to the general belief, most spreadsheets are not one-time models that only exists for a short time. The average lifetime of a spreadsheet is 5 years and during its lifetime they are used on average by 12 persons [37]. Because of their relatively long lifetime, spreadsheets, like software, evolve over time.

Software evolution is a well-researched topic within the domain of software engineering [59], [60], [61]. Research showed that the understanding of source code evolution helps to identify errors within the software and highlights areas that are change-prone and could possibly be improved. Could a better understanding of spreadsheet evolution, bring similar benefits to the domain of spreadsheets? Unfortunately, research on spreadsheet evolution is rather limited and a better understanding of how spreadsheets evolve is needed. We need answers on questions such as:

- How do spreadsheets evolve over time?
- What is the change frequency?
- What kind of changes are made?
- What are the motivations behind these changes?
- Will the study of spreadsheet evolution contribute to better change comprehension and prediction of spreadsheets?

Analyzing the evolution of spreadsheets leads to several challenges. First, we need to be able to correctly identify the changes that were made between the different versions of the spreadsheet. For this study, we are mainly interested in the changes that were made to the formulas. Changes in these formulas change the behavior of the spreadsheet and could possibly introduce errors into the spreadsheet. In this study, we developed FormulaMatch, an algorithm that identifies and visualizes these changes.

The second challenge related to the research of spreadsheet evolution is finding a set of different versions of the same spreadsheet that were developed and maintained over

[1]http://www.eusprig.org/horror-stories.htm

a significant period of time. Dou *et al.* proposed a semi-automated approach to identify evolution groups within a larger spreadsheet group [62]. They applied this approach to the Enron spreadsheet corpus [10] which resulted in a version-ed spreadsheet corpus called VEnron that consists of 7,294 spreadsheets spread over 360 evolution groups. The drawback of this dataset is that we have no access to the creators of the spreadsheet.

In our study, we want to understand the changes and the motivation behind these changes that occur during the evolution of a spreadsheet. Access to the creators is therefore crucial. For this reason, we decided not to use the VEnron corpus, but to cooperate with Alliander. Alliander is a Dutch energy network company. They are responsible for the distribution of electricity and natural gas in a significant part of the Netherlands. They have 5.7 million customer connections, maintain a 90,000 km electricity network and a 42,000 km gas grid and have a yearly revenue of €1.7 billion. Alliander provided us with two sets of spreadsheets that we could use for our analysis.

The remainder of this paper is organized in the following way. In the next section, we provide background information on spreadsheet evolution and discuss related work. In Section 4.2 we discuss the algorithm that we used to detect changes between different versions of a spreadsheet. The two case studies and our findings are presented in Section 4.3. Some topics that affect the applicability and suitability of our findings are discussed in Section 4.4 and we end the paper with concluding remarks in Section 4.5.

4.1 Background & Related Work

4.1.1 Software Evolution

The study of software evolution has provided us with insights that support developers in creating better software. The first studies date back to the late 1960s [63] although the term software evolution itself was not used until 1974 [64]. Lehman and Ramil published a comprehensive summary of 30 years of research on software evolution in 2003 [59].

Research from Novais *et. al.* provides reasons why the study of software evolution matters [60]. They conducted a systematic mapping study of the goals and purposes of software evolution visualization. The 5 most frequently mentioned purposes were: change comprehension, change prediction, contribution analysis, reverse engineering, identification of anomalies, and development communication.

In one of the early studies on software evolution, Gall *et. al.* studied the evolution of the software of a telecommunication switching system over a period of 21 months [61]. The system consisted of about 1500 to 2300 programs. They based their analysis on system properties like size, changing rate, and growing rate. They defined changing rate as the percentage of programs of the system which changed from one release to the next. In a similar manner, growing rate was defined as the percentage of programs of the system which have been added (or deleted) from one release to the next. The method Gall *et. al.* used to study the evolution of the telecommunication switching system could also be used to study the evolution of spreadsheets. Gall *et. al.* analyzed the changes in the system on the level of the individual programs. For spreadsheets, this could be adapted to the level of the individual formulas.

4.1.2 Spreadsheet Evolution

Research on spreadsheet evolution is still limited. In previous work, we compared 54 pairs of spreadsheets [29]. These pairs consisted of the original spreadsheet created by a customer and a version that was rebuilt by professional modelers from the company F1F9. The study provided insight into the effect the rebuilding had on the occurrence of code smells in formulas. However, each spreadsheet was analyzed for only two versions and therefore the obtained insights about the evolution of the spreadsheet were limited.

Most related to our research is the work of Dou *et. al.* [62]. They propose a semi-automated approach to identify evolved spreadsheets and recover the embedded version information. They applied this approach to the Enron corpus [10] [65] and created VEnron, a spreadsheet corpus with version information that consists of 360 evolution groups with a total of 7,294 spreadsheets. An evolution group is a set of spreadsheets that are all originated, either directly or via intermediate versions, from the same spreadsheet. Of these 360 evolution groups, 251 groups, consisting of a total of 4,149 spreadsheets, contained spreadsheets in which formulas were used. The study was mainly focused on identifying evolution groups within the Enron corpus, but the authors also compared the different spreadsheets within an evolution group with Microsoft's Spreadsheet Compare tool and made a technical classification of the type of changes they encountered. They studied 4 types of changes:

- **Structural**: Changes to the structure of the spreadsheet like adding or deleting rows.
- **Entered values**: Changed, added or deleted values (instead of formulas).
- **Formulas**: Changed, added or deleted formulas.
- **Calculated values**: Not a change in the formula, but in the calculated result of the formula, caused by changes in the input values for the formula.

In this paper, we focus on changes in formulas because a change in a formula means a change in the functionality of the spreadsheet. Furthermore, we want to investigate the motivations behind the change. Is it, for example, because of a new feature request, to correct an error or to optimize the performance of the spreadsheet. Dou *et. al.* were not able to answer these questions because they did not have access to the creators of the spreadsheet.

Xu *et. al.* proposed SpreadCluster, a different approach for recovering spreadsheet version information [66]. Instead of clustering spreadsheets based on their filenames, they use features of the spreadsheet, like table headers and worksheet names. Their study shows that SpreadCluster can cluster spreadsheets with a higher precision (78.5% vs. 59.8%) and recall rate (70.7% vs 48.7%) than the filename approach that was used in VEnron [62]. Applying SpreadCluster to the Enron corpus resulted in a new versioned spreadsheet corpus: VEnron2. This study only focused on the clustering of spreadsheets and did not analyze the changes within an identified evolution group of spreadsheets.

4.1.3 Comparing Spreadsheets

Chambers *et. al.* present SheetDiff, a method for identifying the changes between two spreadsheets [67]. They determine all individual cell changes between two versions of the spreadsheet. Next, they optimize the cell changes into higher level changes like adding or deleting a column. As a result, this reduces the number of changes that are presented to the user. In their approach, they detect changes in all cells and not only the formulas.

Inspired by SheetDiff, Harutyunyan *et. al.* developed RowColAlign, an algorithm that can identify differences between two spreadsheets that is based on an algorithm for solving the one dimensional longest common subsequence problem [68]. RowColAlign is very successful in identifying changes caused by row insertion, row deletion, and cell level edits. However, it does not take into account operations such as copy-paste or cell-fill. Furthermore, the algorithm has not been applied to spreadsheets with cells containing formulas.

4.1.4 Spreadsheet Evolution Challenges

Studying spreadsheet evolution comes with its own challenges. It starts with the lack of version control systems (VCS). Spreadsheet users do not use Github or SVN because VCS are in general not suited to store the version history of spreadsheets. The lack of version control systems, also means that there are no commits and no commit messages. As a result, it is challenging to find a group of spreadsheets that belong to the same overall project. If such a group is found the next challenge is to determine the exact order in which they were created.

The next step in analyzing the evolution of spreadsheets is comparing the different versions of a spreadsheet in an evolution group. Finding changes between two versions of a program is relatively straightforward. Most programming languages are text-based and programs, classes and methods are organized in text files. Modern version control systems provide 'Diff' tools to highlight the differences between two versions on the level of code lines or even words.

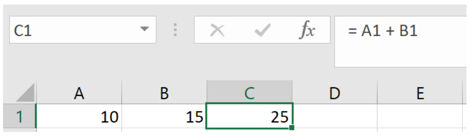
A normal 'Diff' tool is not able to compare two spreadsheets. Spreadsheets are not text-based files. A workbook consists of worksheets and worksheets consists of cells. The information in a spreadsheet is entered in a cell. To track changes in a spreadsheet you have to track changes on cell level. But what if a user inserts a row or a column or both. How should a compare tool determine which cells should be compared to each other?

Another important difference between source code and spreadsheets is the combination of data and code. In a spreadsheet data and code exist next to each other. If a user changes a numeric value in a single cell then all formulas that make reference to this cell will calculate a different outcome. If you compare the spreadsheet these changes will be detected. However, no change was made to any formula and one could argue that the spreadsheet is still unchanged.

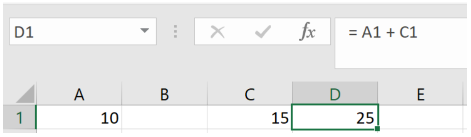
An additional problem is the way the structure of a spreadsheet influences the formulas. The example of Figure 4.1 elucidates this problem.

Figure 4.1a shows a formula in cell C1 that adds up the values in cell A1 and B1, resulting in a calculated value of 25. Assume a user inserts a new column after column A. The result is shown in Figure 4.1b. Because of a structural change in the worksheet (adding a new column) the formula has changed. However, the function of this formula is still unchanged, as is its calculated value.

Now imagine that this spreadsheet contained 1,000 rows and that the formula in cell C1 of Figure 4.1a was copied down to all 1,000 rows. In that case, a single insertion of a



(a) Original formula



(b) Changed formula after inserting column B

Figuur 4.1: The effect of changes in structure on formulas

new column would result in 1,000 formula changes. If these were all taken into account while comparing the two versions of the spreadsheet, the number of reported changes would be overwhelming and analysis of the differences between the two versions would be challenging.

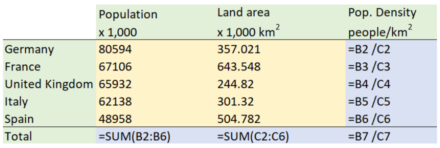
4.2 Detecting Changes

As stated in the Section 4.1, analyzing changes between two spreadsheets is difficult. Simple structural changes to a worksheet, like inserting a row or a column, can lead to a myriad of changes (see Figure 4.1). The goal of our study is to understand how spreadsheets evolve. We are especially interested in how formulas change over time. The formulas in a spreadsheet can be compared with the source code of a program. It determines its functionality, and it is in the formulas where most errors emerge. Furthermore, one would not argue that a program has been changed when it is run for a second time with different data. The same holds true for spreadsheets. If just the data has been changed the spreadsheet should be considered as unchanged.

The number of detected changes reduces if only changes in the individual formulas are taken into account and all changes in structure, data and formatting are ignored. However, it still can result in thousands of changes. More complex spreadsheets contain a lot of formulas. For example, one of the models that we used in our case studies contained about 100,000 formulas. In such a spreadsheet, as described in the previous Section, a single change like moving a cell, can lead to a change in thousand related formulas. Presenting all these changes to the user, will not help them to understand the risks induced by the applied changes.

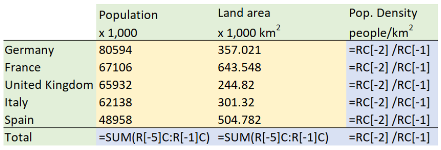
4.2.1 Unique Formulas

For this reason, the evaluation is reduced to only the unique formulas. We make use of the R1C1 notation of a formula to detect the unique formulas in a spreadsheet [17]. In Figure 4.2 we illustrate this with an example. In Figure 4.2a we see a small spreadsheet with 8 formulas. None of these formulas are the same. However, if we switch in Figure 4.2b from A1 notation (meaning the cell on the first row and the first column is referred to as A1) to the R1C1 notation (meaning that same cell is now referred to as R1C1) most of the formulas are the same. It turns out, as highlighted in Figure 4.2c, that there are actually only two unique formulas.



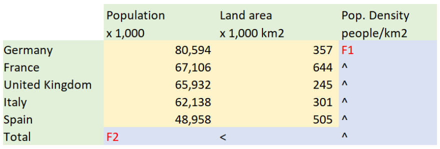
	Population x 1,000	Land area x 1,000 km^2	Pop. Density people/km^2
Germany	80594	357.021	=B2 /C2
France	67106	643.548	=B3 /C3
United Kingdom	65932	244.82	=B4 /C4
Italy	62138	301.32	=B5 /C5
Spain	48958	504.782	=B6 /C6
Total	=SUM(B2:B6)	=SUM(C2:C6)	=B7 /C7

(a) Formulas in A1 notation



	Population x 1,000	Land area x 1,000 km^2	Pop. Density people/km^2
Germany	80594	357.021	=RC[-2] /RC[-1]
France	67106	643.548	=RC[-2] /RC[-1]
United Kingdom	65932	244.82	=RC[-2] /RC[-1]
Italy	62138	301.32	=RC[-2] /RC[-1]
Spain	48958	504.782	=RC[-2] /RC[-1]
Total	=SUM(R[-5]C:R[-1]C)	=SUM(R[-5]C:R[-1]C)	=RC[-2] /RC[-1]

(b) Formulas in R1C1 notation



	Population x 1,000	Land area x 1,000 km2	Pop. Density people/km2
Germany	80,594	357	F1
France	67,106	644	^
United Kingdom	65,932	245	^
Italy	62,138	301	^
Spain	48,958	505	^
Total	F2	<	^

(c) Visualization of unique formulas

Figuur 4.2: Detecting Unique Formulas

4.2.2 Similarity score

When comparing two spreadsheets we start with detecting the unique formulas. For every formula in the first version V0, we have to find the equivalent in the second version V1. It is not possible to use the cell address, because it could have been changed if rows or columns were added or deleted. Furthermore, we have to take into account that the formula itself could have been changed.

To find a matching formula we analyze the similarity of the formulas in their R1C1 notation. As a measure of similarity we use the Levenshtein Distance [69], [70]. It denotes the minimum number of operations needed to transform one string into the other. A larger distance means that the strings are less similar.

With this similarity measure, we can match the formulas in V0 with the formulas in V1 that have the highest similarity. Still, this does not guarantee a perfect match, as different formulas can have the same similarity score. To overcome this the formulas are also compared to the following additional properties:

- Calculated result: the outcome of the formula.
- Parse tree depth: This is a measure of how nested a formula is.
- Path depth: a formula in a spreadsheet can make reference to another formula which again can make a reference to another formula. The path depth is the length of the complete calculation chain.
- Direct dependents: the number of cells that make a reference to the outcome of the formula.
- Direct precedents: the number of cells that are used by the formula as input.

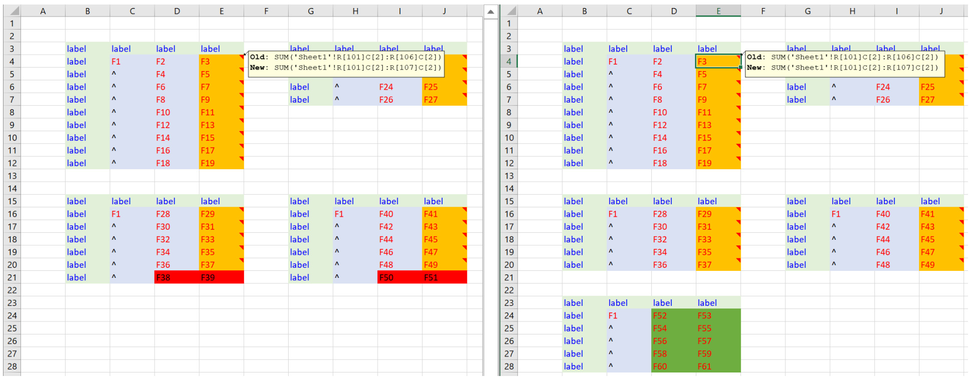
4.2.3 Matching Algorithm

Trying to match the formulas in V0 with the formulas in V1, based on the maximum similarity score while taking into account the additional properties gives good results when the number of formulas in both versions are the same and no formulas have been added or deleted. To correctly handle the addition or deletion of formulas a more sophisticated matching algorithm is needed.

In their paper “College Admissions and the Stability of Marriage” [71], Gale and Shapley describe an easy and elegant algorithm that solves the problems caused by the adding and deleting of formulas. They describe an algorithm that solves the so-called College Admission problem. In this problem, colleges are considering a set of n applicants. They can only accept a quota of q applicants. Of course, applicants want to be accepted at the college of their preference. Gale and Shapley designed an algorithm that will lead to an optimal stable assignment, meaning that: *“every applicant is at least as well off under it as under any other stable assignment”* [71].

To accomplish this, every college should make an ordered list of their preferred applicants and every applicant should make an ordered list of their preferred colleges. Then the algorithm can be applied as follows:

1. All applicants apply to the college of their first choice.
2. A college with a quota of q places q highest ranking applicants on its waiting list and rejects the rest.
3. Rejected applicants then apply for their second choice.
4. Each college selects the top q from the new applicants and those on its waiting list, put these on its new waiting list and rejects the rest.
5. This is repeated until all applicants are either on a waiting list or have been rejected by all of their preferred colleges.



```
Old: SUM('Sheet1'!R[101]C[2]:R[106]C[2])
New: SUM('Sheet1'!R[101]C[2]:R[107]C[2])
```

Figuur 4.3: Visualization of FormulaMatch

We applied this idea to develop FormulaMatch, an algorithm that we can use for our matching problem. For each formula in V0 (*’the colleges’*) we created an ordered list of preferred formulas in V1 (*’the applicants’*) using the similarity score. Also for all formulas in V1, we created an ordered list for their preferred formulas in V0. Every formula in V0 can only be matched to exactly one formula in V1, so we set the quota of the V0 formulas to 1. Then we applied the steps of the algorithm.

The result of FormulaMatch provides us for each formula in V0 with a matching formula in V1. Based on the similarity scores we know if the formula was changed or not. It is possible that no matching formula was found, meaning that this formula was deleted in V1. It is also possible that for a formula in V1 no matching formula in V0 was found. In terms of the algorithm, it was rejected. Rejected formulas are not existing in V0 and therefore, were added in V1.

Figure 4.3 shows a visualization of FormulaMatch[2]. All unique formulas are indicated with an identifier (FXX). If they were changed, they are highlighted in orange and in a comment the old and new version of the formula in R1C1 notation is displayed. Deleted formulas are highlighted in red and newly created formulas in green.

4.3 Spreadsheet Evolution in two Industrial Case Studies

4.3.1 Setup

The goal of this paper is to obtain a better understanding of spreadsheet evolution. With the results of the case studies we will answer the following research questions:

RQ1 How do spreadsheets evolve over time?

RQ2 How common are changes in formulas during the life-span of a spreadsheet?

RQ3 What are the reasons behind the changes?

RQ4 To what extent can the results of a spreadsheet evolution study, support end-users in creating spreadsheets that are easier to maintain and contain fewer errors?

For the case studies, we cooperated with Alliander. Alliander is one of the large energy network companies in the Netherlands and we had the opportunity to work with employees from the Analytics group of the Asset Management department. They provide data-based insights from maintenance and failure reports to support the development and maintenance of the energy network. We asked them to provide us with examples of spreadsheets that they created and maintained for several years. Based on this request we received two spreadsheet evolution groups that together contained a total of 73 spreadsheets. Before we started the case studies we provided the owners of the spreadsheets with information about the setup of the study and they gave us a short explanation about the purpose and context of their spreadsheets.

In the design of our study, we followed a mixed methods approach [72]. We started with a quantitative phase that consisted of a detailed analysis of the evolution of the received spreadsheets and followed up with a qualitative phase that consisted of interviews with the creators of the spreadsheets.

During the two case studies, the procedure was as follows: First, we asked the creators of the spreadsheet to put the spreadsheets in the evolution group in the correct update

[2]To protect the sensitive data in the spreadsheet, all column and row headers in the example have been replaced with the term 'label'

order. Subsequently, we analyzed the spreadsheets with our Spreadsheet Scantool and ran for each pair of consecutive versions of the spreadsheets our FormulaMatch algorithm. Based on the outcome of this analysis we summarized the evolution of the spreadsheet with the below-mentioned metrics. These metrics are the same as used by Gall *et. al.*[61], but we have adjusted them in the following way to make them suitable for the use with spreadsheets.

- Size: As stated in Section 4.1, data and code are combined in a spreadsheet. The code in a spreadsheet is formed by the formulas and the number of unique formulas (see Section 4.2.1) is used in this study, as metric for size. Still, formulas are not the only component of a spreadsheet that is responsible for its growth. A spreadsheet can also grow in size by just adding data to it. Therefore, we also use the number of non-empty cells as a metric for size.
- Changing rate: is defined as the percentage of unique formulas in the spreadsheet that changed from one version to the next.
- Growing rate: is defined as the percentage of unique formulas that have been added (or deleted) from one version to the next.

Subsequently, we used the results from the FormulaMatch algorithm to manually inspect all changes that occurred in the unique formulas. After that, we interviewed the creators of the spreadsheets and asked them the following questions:

1. How would you describe the development process of the spreadsheet?
2. Which parts in the spreadsheet do you consider complex?
3. Which important changes that you made in your spreadsheets do you remember?
4. What were the main reasons behind the changes you made in the spreadsheets?

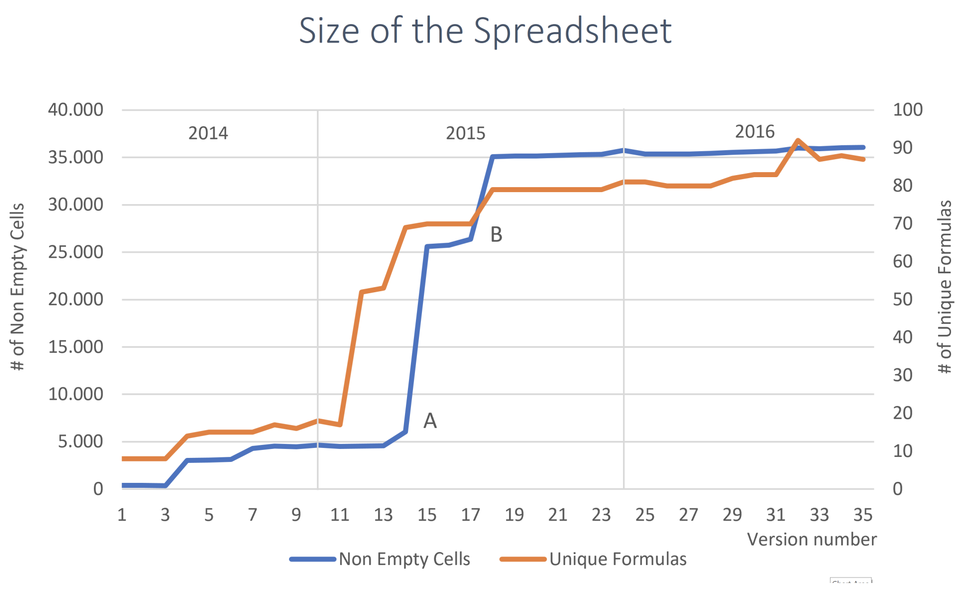
The answer to these questions, combined with the data collected from the analysis of the spreadsheets, provides more background and context of the evolution process and will enable us to answer the research questions.

After these questions we discussed the findings of our analysis with the creators, to get a better understanding of the motivations behind some of the changes. We also asked them if and how the results of the study could help them to create and maintain better spreadsheets.

4.3.2 Case Study I: failure density in the natural gas distribution network

The first case study concerns a failure density model for the gas distribution network. It reports monthly on failure incidents, causes of failures and it provides a forecast for the coming months of the year. We received 35 spreadsheets that span a period of almost 3 years. The first spreadsheet dates back to February 2014 and the last version is from January 2017.

Figure 4.4 shows the evolution of size in number of cells over the different versions of the spreadsheet. As stated in Section 4.1, spreadsheets consist of data and code. When



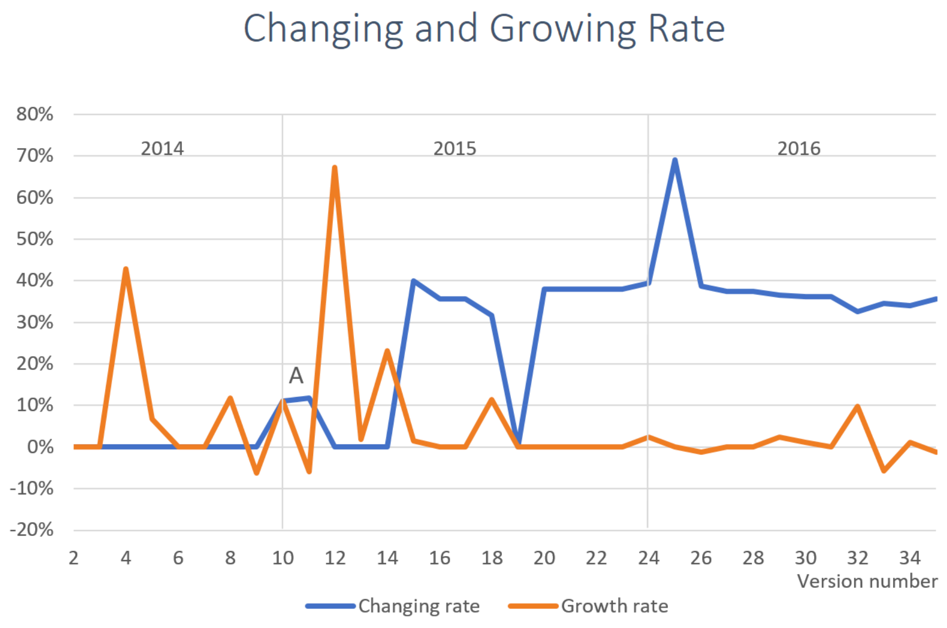
Figuur 4.4: Evolution of size of the spreadsheet over several versions

analyzing the size of the spreadsheet, a distinction has to be made between data and formulas. Therefore, in the chart we show the number of non-empty cells as a metric for the size of the data and the number of unique formulas as metric for the formulas. The spreadsheet starts in version V01 with 383 non-empty cells and grows in V35 to 36,079 cells. The unique formulas grow from 8 in V01 to 87 in V35. These 87 unique formulas represent a total of 2,046 formulas.

Although the growth of the data and the unique formulas follow a comparable pattern, there is no relation between the two. During the interview, we discussed the growth patterns with the creator of the spreadsheet. The two steep increases (A and B in Figure 4.4) of non-empty cells were related to adding and extending a reference table with zip code information of the Netherlands. The increases in unique formulas that occurred at almost the same moments were related to new information requests and were not related to the zip code table.

Figure 4.5 displays both the changing and growing rate of the spreadsheet. In the early versions, the model was extended with new functionality. We see this between V03 and V04 and around V11 to V14. This corresponds to the growth of the unique formulas shown in Figure 4.4. During this time frame, new functionality was added to the model based on requests from end-users for more detailed information. At the same time, there were almost no changes. Only a small spike around V10 and V11 (See A in Figure 4.5) with a changing rate of 10%. These changes coincide with a year-end rollover. Something similar can be seen a year later from version 24 to 25. Some formulas needed to be changed when the model was transferred from one year to another.

Until version 14 there are, except for the year-end rollover, no changes in the existing formulas. This changed in version 15, from that moment the changing rate continuously



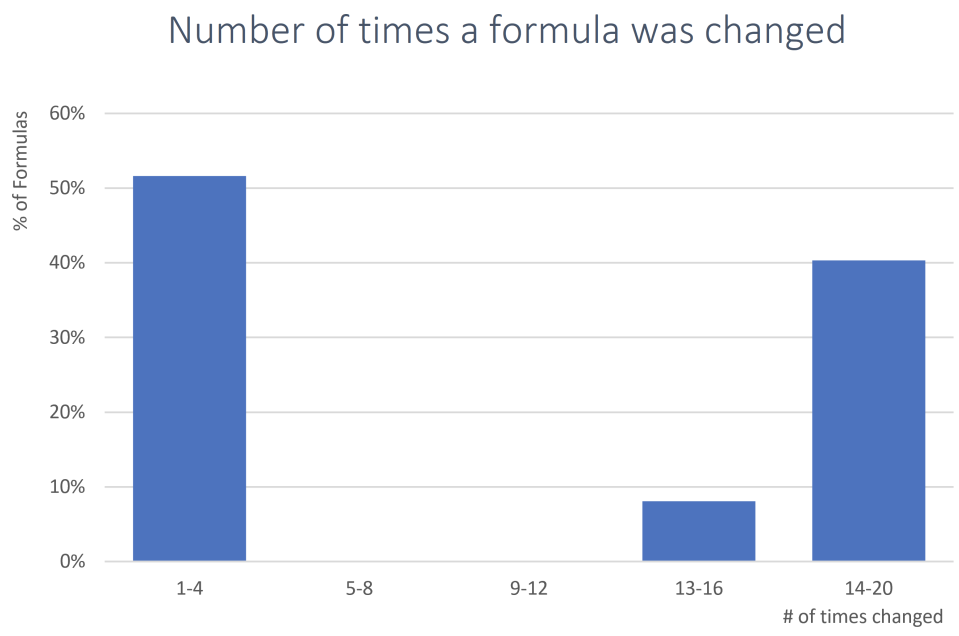
Figuur 4.5: Changing and growing rate of the spreadsheet over several versions

stays at a level of about 40%. We discussed this with the creator of the spreadsheet. In V14 a set of formulas were added that needed to be adjusted every time data for a new month was added. Therefore, it stands out that in V19 the changing rate suddenly drops back to 0%. In this version, new data for the month was added, but the adjustments in the formulas were forgotten.

In the 35 versions of the spreadsheet, 617 changes to unique formulas were detected. These 617 changes were made to only 62 unique formulas, meaning that a lot of unique formulas were changed multiple times. Figure 4.6 shows the frequency distribution of these changes. About half of the formulas were changed between one and four times, the other half between thirteen and twenty times. This relates to the set of formulas that had to be changed every month. It started in V15 with 25 formulas and was later, in V20, extended to 30 formulas.

During the interview, the creator of the spreadsheet summarized the changes that were made in the 35 different versions as providing gradually more information to the end-users of the spreadsheet. The spreadsheet is part of a monthly reporting cycle and by providing more information a lot of recurring questions could already be answered based on the information in the spreadsheet. The change that was remembered the most coincides with the spike in the changing rate in V24 (see Figure 4.5).

According to the creator of the spreadsheet, the most complex part of the spreadsheet are the formulas that calculate a forecast for the coming months. From the 617 changes, only 3 were related to this part of the spreadsheet. In this case, the complexity of a formula was not a driver for frequent change.



Figuur 4.6: Frequency distribution of formula changes

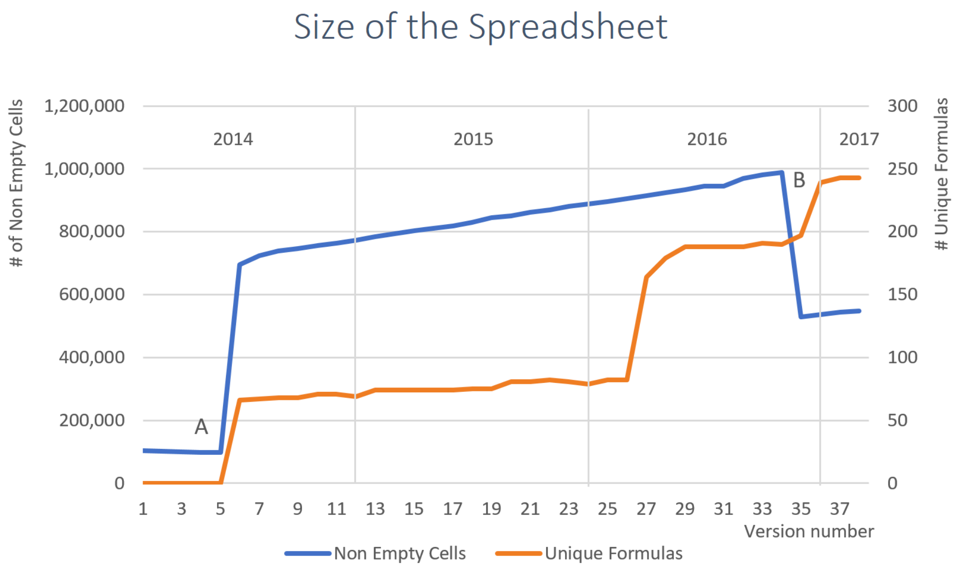
The main reasons for changing formulas in the spreadsheet, mentioned during the interview, are 1) incorporating new requests from the end-users of the spreadsheet or 2) changing formulas to make future updates and maintenance easier. The creator of the spreadsheet is well aware of the fact that there are areas in the spreadsheet that could be improved. It should not be necessary to update formulas every time data is added to the spreadsheet. However, it is not always easy to find the time to implement such changes.

4.3.3 Case Study II: Failure analysis of the Medium Voltage Grid

The second case study is a set of spreadsheets providing Alliander with an analysis and forecast of failures in the Medium Voltage Electricity Grid. The set consists of 38 spreadsheets with a time span of 38 months, from January 2014 to February 2017.

The evolution of size for the complete set of spreadsheets is shown in Figure 4.7. The data component in this spreadsheet is much larger in comparison with the spreadsheets from the first case study. The chart indicates two moments (A and B in Figure 4.7) where the number of non-empty cells suddenly changes. In June 2014 the number of cells changes from 99,091 cells to 695,471. In the interview, we asked the creator about the reason for this change. Until that time the analysis was based on the last twelve months of data. As from June 2014, this was extended to a time span of five years.

In November 2016 a large part of the data is deleted, reducing the size of the model to 529,853 cells. First, we assumed that old data was removed from the model. We discussed this during the interview and it turned out that this was only a partial explanation of the size reduction. Indeed one year of data was deleted but at that time the model contained



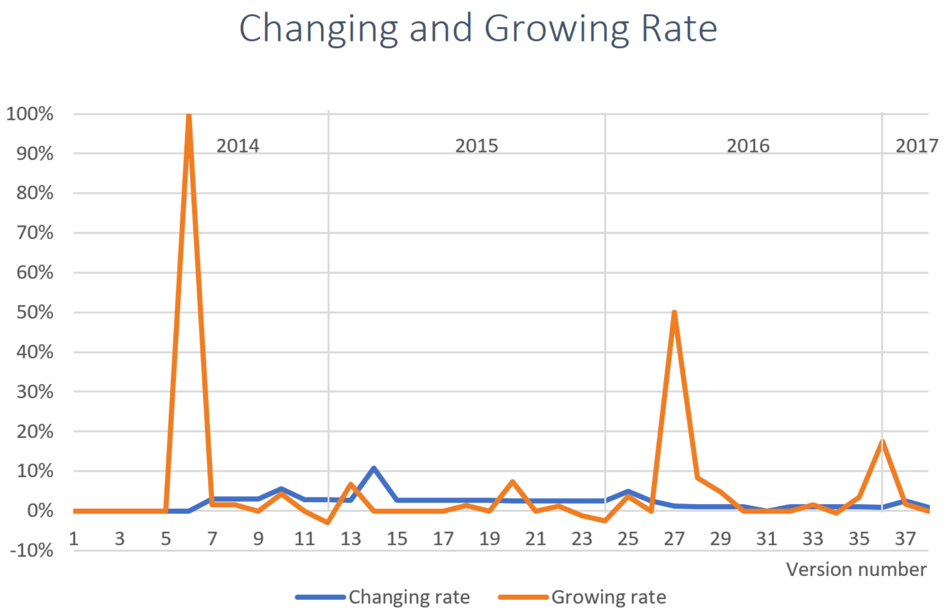
Figuur 4.7: Evolution of size of the spreadsheets over several versions

almost one million non-empty cells and it started to get really slow. Closer inspection of the source data revealed that it was possible to reduce it by filtering out rows with a status that was not needed for the calculations. It was this filtering that was mainly responsible for the size reduction.

The model started with one pivot table and without formulas. in Figure 4.7 we can see that there were three distinctive moments that new unique formulas were added to the model: mid-2014, early 2016 and late 2016. In all cases, the reasons behind the addition of new formulas were requests for more insights from the end-users of the spreadsheet. The ratio of unique formulas to formulas is about 1 to 1,000, which is much higher than in the first case study where it was about 1 to 25. This high ratio can be explained by the fact that the model contains a data set of about 10,000 rows and that formulas are defined for each individual row. This means one unique formula with 9,999 siblings.

The changing and growing rate are shown in Figure 4.8. The three moments when formulas were added to the model are clearly recognizable. In comparison with the first case study, this model is much more stable. The changing rate stays below ten percent for most of the time. However, also in this model, there are formulas changed in almost every version. Figure 4.9 illustrates this.

In the 38 versions of the spreadsheet 76 changes were made to only 18 unique formulas. The majority of these formulas were only changed once but there are four formulas that were changed multiple times. We discussed this in the interview with the creator of the spreadsheet. There were two formulas that had to be changed every month. The range that was used in these formulas to calculate a forecast had to be adjusted. The other two formulas had to change only once a year. The logic for the calculation of a KPI was different in the last month of the year than in the other months. Finally, we saw that at the end of



Figuur 4.8: Changing and growing rate of the spreadsheets over several versions

2016 a set of four new formulas were added that had the year hardcoded in the formula. These formulas had to be changed at the beginning of 2017.

This model started with one pivot table. When we asked the creator of the spreadsheet to summarize the development of the model in the 38 consecutive versions, he explained that most changes had to do with new information requests from the end-users of the spreadsheet, but also to replace the pivot table with formulas to mitigate the risk that the spreadsheet was updated without updating the pivot table and therefore presenting information that was outdated.

According to the creator of the spreadsheet, the most complex part of the model was related to the calculation of the long-term and short-term trend of the failure analysis. Although complex, the formulas used for this calculation were stable. They were not changed in any of the 38 spreadsheets.

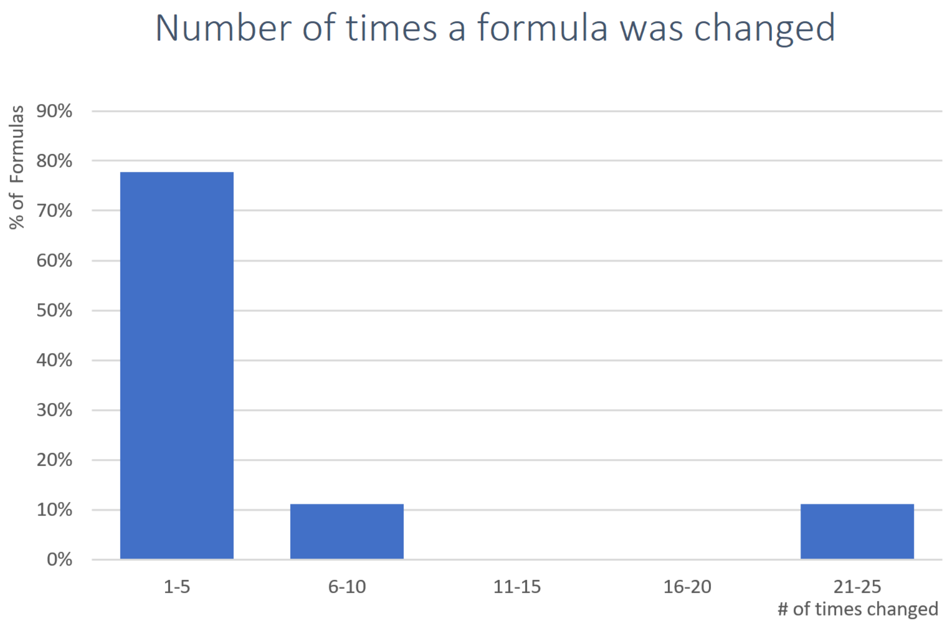
The changes that were remembered the most by the creator of the spreadsheet, coincide with the three spikes in the growth rate in Figure 4.8.

The creator of the spreadsheet was well aware of the formulas that needed maintenance every month and also possesses the knowledge of how to change them. Finding the time and priority is, in this case, the limiting factor.

4.3.4 Conclusion

Based on the information collected during the case studies, we revisit the research questions.

RQ1 How do spreadsheets evolve over time? In both cases, it is clear that the



Figuur 4.9: Distribution of change frequency of formulas

spreadsheets grow over time, both in the number of non-empty cells as unique formulas. It also became clear that, for the cases considered, there is no direct relation between the growth in data and the growth in unique formulas. Spreadsheets that exists for a longer time-span are often used for reporting purposes. The growth of the data is caused by adding more data points to the analysis. Growth in unique formulas is caused by adding new functionality to the spreadsheet.

RQ2 How common are changes in formulas during the life-span of a spreadsheet? For both cases, unique formulas were changed in almost every version. Only the number of changes differed between the two cases. The addition of new data points made it necessary to change the logic of some of the formulas. In the context of a spreadsheet this sounds reasonable but translated to the context of software evolution it would mean that source code need to be changed every time the program is run with new data.

RQ3 What are the reasons for the changes? The motivation for most changes in the formulas is new feature requests from the end-users of the spreadsheet. Another reason for change is to improve the maintainability of the spreadsheet. A third reason, that was not mentioned during the interviews, but of which we found examples in our evolution study, is the correction of an error that was made in a previous version.

RQ4 To what extent can the results of a spreadsheet evolution study, support end-users in creating spreadsheets that are easier to maintain and contain fewer errors? When we discussed the results of the evolution study with the creators of the spreadsheet, several suggestions were made about how these results could support a spreadsheet user in making a better model:

- Summarize the changes that are made in a new version. It helps the creator of the spreadsheet to detect if all changes are intended and correctly implemented. Choosing to only show the changes in unique formulas instead of all formulas helps to present the changes in a concise way to the creator of the spreadsheet.
- Provide the creator of the spreadsheet with a list of formulas that were frequently changed in earlier versions of the spreadsheet. This could function as a checklist to make sure that all necessary changes have been made
- Suggest formulas that are candidates for refactoring. Formulas that have to be changed in every new version can often be rewritten in such a way that changes are not necessary.
- Highlight sudden drops or spikes in changing and growing rates. They sometimes indicate anomalies in the spreadsheet. For example, a sudden drop in the changing rate from 32% to 0% between V18 and V19 in case study I revealed a set of formulas that should have been updated but were not.

4.4 Discussion

4.4.1 Threats to validity

The cooperation with Alliander gave us a unique opportunity to study spreadsheet evolution in real-life scenarios. However, a real-life dataset comes with the price of reduced repeatability. We strongly support open data, but because the spreadsheets contain sensitive information, we are not able to share them.

A threat to the external validity of our study concerns the representativeness of the selected set of spreadsheets that we analyzed in this paper. The aim of this study is to gain a better understanding of the evolution of spreadsheets. The direct access to the creators of the spreadsheet is of decisive importance to obtain this understanding, but also sets a practical limit on the number of spreadsheets that could be included. However, if we look at properties, like size and number of formulas, of the spreadsheets then they are comparable to the spreadsheets in the Enron corpus [10] and with this respect, they seem to be representable.

4.4.2 VBA Code, Pivot Tables, and Charts

In this study, we have chosen to study the evolution of spreadsheets by analyzing the changes in formulas. They determine the functionality of the spreadsheet and it is in the formulas that most errors occur. However, there are other components in spreadsheets that evolve over time, such as pivot tables, charts, and VBA code. In future research, we will shift our focus to these components.

4.5 Concluding Remarks

The aim of this paper is to get a better understanding of the evolution of spreadsheets. In the two case studies, we saw that spreadsheets grow over time, both in data as in the number of formulas. The main drivers for the growth of the formulas are new feature requests from the end-users of the spreadsheet. Besides new feature requests also improving the maintainability was mentioned as a motivation to implement changes and finally, we saw that some changes were related to bug fixing. In both case studies, there was a certain percentage of formulas (48% in study I and 22% in study II) that changed in almost every version. Results show that these formulas had to be adjusted when new data points were added to the spreadsheet.

The contributions of this paper are as follows:

- FormulaMatch: an algorithm to match unique formulas of two different versions of the same spreadsheet (Section 4.2).
- Two case studies in which a detailed study is made of the evolution of a spreadsheet. In both cases, the analyzed spreadsheets had a time-span of three years (Section 4.3).
- Insights from spreadsheet users about how the results of an evolution study can support them in creating better spreadsheets (Section 4.3).

This research gives rise to several directions for future work. The FormulaMatch algorithm in combination with the concept of unique formulas makes it possible to present in a concise way the changes between two versions of a spreadsheet. We will research if it is possible to further improve the results by combining the FormulaMatch algorithm with other algorithms that detect structural changes in spreadsheets. Furthermore, the case studies showed that measuring the change frequency of formulas over the life-span of a spreadsheet supports 1) the identification of formulas that are good candidates for refactoring or 2) predicting which formulas need to be changed in the version on hand. We are planning to develop a tool that uses this change frequency analysis to present this information to the user. Finally, we will study the evolution of VBA code, pivot tables, and charts within a spreadsheet.

5 XLBlocks: a Block-based Formula Editor for Spreadsheet Formulas

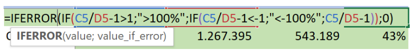
Spreadsheets are frequently used in industry to support critical business decisions. Unfortunately, they also suffer from error-proneness, which sometimes results in costly consequences. Experiments in the field of program education have shown that programmers tend to make fewer errors and can better focus on the logic of a program if they use a block-based language instead of a textual one. We hypothesize that a block-based formula editor could support spreadsheet users in a similar way. Therefore, we develop XLBlocks and conduct a think-aloud study with 13 experienced spreadsheet users from industry. Participants are asked to create and edit several formulas, using our block-based language. We then ask them to evaluate this editor using the Cognitive Dimensions of Notations framework. We found that for all dimensions the block-based formula editor received a better evaluation than the default text-based formula editor.

This chapter has been published as *B. Jansen and F. Hermans. XLBlocks: a Block-based Formula Editor for Spreadsheet Formulas, VL/HCC 2019* [31]

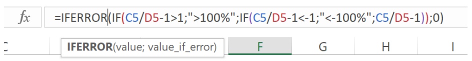
Spreadsheets are widely used in industry. It has been estimated that 90% of all analysts in industry use spreadsheets for their calculations [2]. This observation indicates that spreadsheets are often used to support critical decisions. Unfortunately, almost all spreadsheets contain non-trivial errors[11]. As a consequence, decisions are based on incorrect information, which eventually can lead to significant loss of money[1]. To address these quality problems in spreadsheets, Hermans *et al.* researched the concept of code-smells in spreadsheet formulas and worksheets [28],[27]. That work was later extended by Cunha *et al.*[38]. Barowy *et al.* extended the concept of smells to also cover data [58].

Concerning errors, Panko and Halverson distinguish three types of spreadsheet errors: mechanical (typing errors), logic (formula errors), and omission errors [73]. Panko also found that error rates for logic errors are higher than for mechanical errors, meaning, that most spreadsheet errors have their origin in formulas [7]. Therefore, although a spreadsheet model consists of data, layout, and formulas, we focus in this paper on creating and maintaining formulas in spreadsheets.

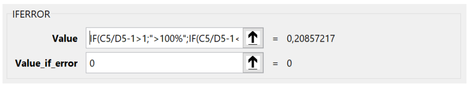
Unfortunately, the way the user interface facilitates the entering of formulas is rather limited. For example, in Microsoft Excel, formulas can be entered directly in a cell (see Fig. 5.1a), in the formula bar (see Fig. 5.1b), or created using the function wizard (see Fig. 5.1c).


=IFERROR(IF(C5/D5-1>1;">100%";IF(C5/D5-1<-1;"<-100%";C5/D5-1));0)
IFERROR(value; value_if_error) 1.267.395 543.189 43%

(a) In-cell editing


=IFERROR(IF(C5/D5-1>1;">100%";IF(C5/D5-1<-1;"<-100%";C5/D5-1));0)
IFERROR(value; value_if_error) F G H I

(b) Editing in the formula bar


IFERROR
Value IF(C5/D5-1>1;">100%";IF(C5/D5-1< = 0,20857217
Value_if_error 0 = 0

(c) Using the function wizard

Figuur 5.1: three ways for entering a formula in Excel

In all 3 of the editing methods, it is difficult to get an overview of the complete formula. For example, the double nested *if* formula in Figure 1.2b is represented as a string of characters on a single line. It is difficult to distinguish the two *ifs* from their arguments. The function wizard attempts to visualize this better but has the drawback that it only shows one function at a time. In Figure 5.1c the *iferror* functions is highlighted, but the *ifs* are 'hidden' in the value field of the *iferror* function. Especially with in-cell editing or using the formula bar, it is easy to forget or misplace a comma, parenthesis, or quote. In these cases, the user needs to know the exact syntax of the function, meaning, the order and purpose of the function's arguments and whether they are mandatory or optional.

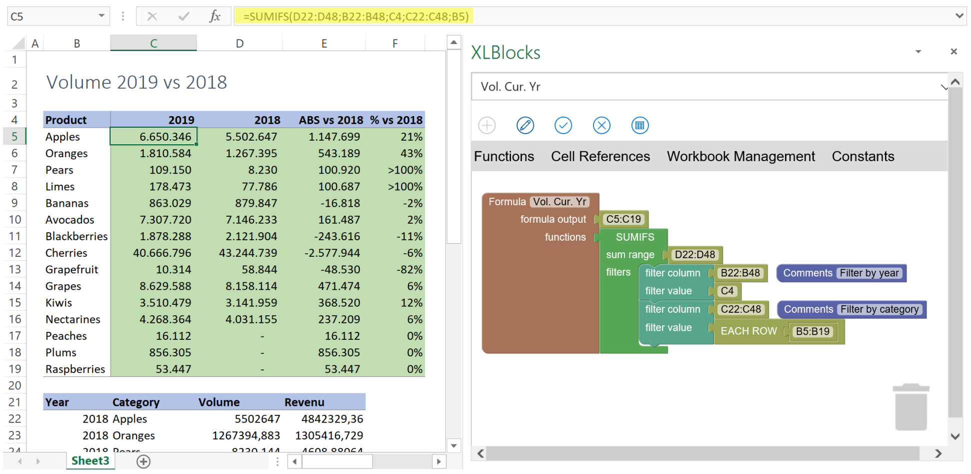
These problems with formula syntax are similar to the challenges novice programmers encounter when they learn to program. Research has shown that block-based program languages improve the performance of novice programmers by minimizing the possibility of syntax errors and removing the necessity for accurate punctuation [74, 75].

[1]http://www.eusprig.org/horror-stories.htm

We hypothesize that a block-based formula editor for spreadsheets could support spreadsheet users in a similar way. This paper introduces XLBlocks, a block-based formula editor for Excel and presents the results of a think-aloud study in which the participants perform a set of typical spreadsheet task with XLBlocks. After the tasks, we interview them and ask them to evaluate XLBlocks using the Cognitive Dimensions of Notation (CDN) framework [26].

5.1 XLBlocks: a block-based formula editor

This paper examines to what degree a block-based formula editor could support professional spreadsheet users while developing or maintaining formulas. In this section, we provide an overview of XLBlocks (Figure 5.2). XLBlocks aims to 1) implement a block-based interface (Section 5.1.1), 2) facilitate users to replicate formulas across rows or columns (Section 5.1.2), and 3) introduce new functions that are easier to use than some of the built-in Excel functions (Section 5.1.3).



C5 =SUMIFS(D22:D48;B22:B48;C4;C22:C48;B5)

Volume 2019 vs 2018

Product	2019	2018	ABS vs 2018	% vs 2018
Apples	6.650.346	5.502.647	1.147.699	21%
Oranges	1.810.584	1.267.395	543.189	43%
Pears	109.150	8.230	100.920	>100%
Limes	178.473	77.786	100.687	>100%
Bananas	863.029	879.847	-16.818	-2%
Avocados	7.307.720	7.146.233	161.487	2%
Blackberries	1.878.288	2.121.904	-243.616	-11%
Cherries	40.666.796	43.244.739	-2.577.944	-6%
Grapefruit	10.314	58.844	-48.530	-82%
Grapes	8.629.588	8.158.114	471.474	6%
Kiwis	3.510.479	3.141.959	368.520	12%
Nectarines	4.268.364	4.031.155	237.209	6%
Peaches	16.112	-	16.112	0%
Plums	856.305	-	856.305	0%
Raspberries	53.447	-	53.447	0%

Year	Category	Volume	Revenu
2018	Apples	5502647	4842329,36
2018	Oranges	1267394,883	1305416,729

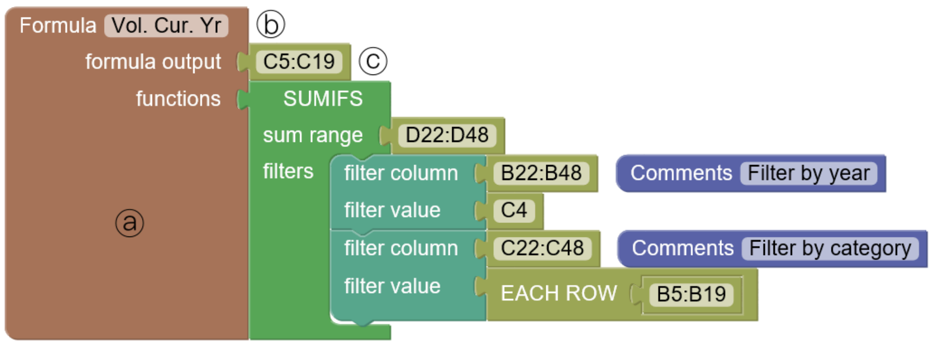
Figuur 5.2: Left the traditional screen with the textual formula highlighted at the top, right the block representation of the same formula

5.1.1 XLBlocks Interface

XLBlocks is an Excel Add-in developed with the Excel JavaScript API [76]. The Blockly library [77] was extended with custom blocks and a code generator for the definition and generation of spreadsheet formulas. Based on frequently used functions in the Enron corpus [10], we included the following functions in the research prototype: SUM, SUMIFS, IFERROR, INDEX, MATCH, VLOOKUP, IF, -, /, >, <. Videos demonstrating the user interface of XLBlocks are available online[2].

To use XLBlocks, a user starts the definition with a formula block (see (a) in Figure 5.3). The user can give the formula a name (b), has to specify the output range (c) (the

[2]https://doi.org/10.6084/m9.figshare.8863532

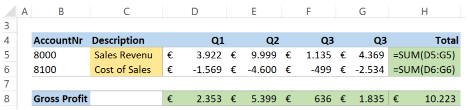


Figuur 5.3: Example of a block definition of a SUMIFS formula

cells in the spreadsheet that will receive this formula) and the functions that are used in the formula. In Figure 5.3 the SUMIFS function is used as an example. This function will calculate a conditional sum based on one or more criteria. Therefore, in XLBlocks, it is possible to connect one or more filter blocks as arguments to the SUMIFS function. In this example, two comments have been added to the formula to document the filters that are used in this sum. These comments are not transferred to the spreadsheet but will be available in the XLBlocks editor.

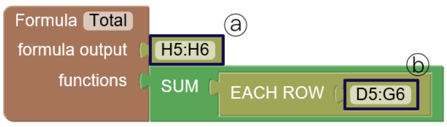
5.1.2 EACH ROW and EACH COLUMN

Within the spreadsheet paradigm, it is common practice to define a formula and then replicate it across many rows or columns by dragging it down or to the right. To facilitate this in the XLBlocks editor, we introduced two special blocks: 'EACH ROW' and 'EACH COLUMN'.



	A	B	C	D		E		F		G		H
3												
4		AccountNr	Description		Q1		Q2		Q3		Q3	Total
5		8000	Sales Revenu	€	3.922	€	9.999	€	1.135	€	4.369	=SUM(D5:G5)
6		8100	Cost of Sales	€	-1.569	€	-4.600	€	-499	€	-2.534	=SUM(D6:G6)
7												
8		Gross Profit		€	2.353	€	5.399	€	636	€	1.835	€ 10.223

(a) Row totals in excel



(b) EACH ROW block in XLBlocks

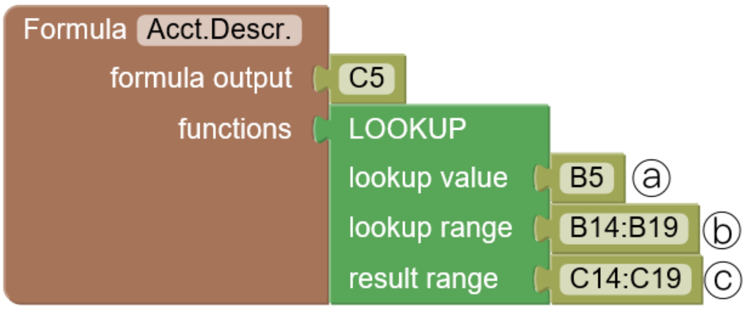
Figuur 5.4: Using EACH ROW to replicate formulas across multiple rows

Figure 5.4 shows an example of calculating row totals in cells H5 and H6. In Excel, a user would define the formula in cell H5 and copy it to H6. In XLBlocks it is sufficient to specify both cells H5 and H6 as output range (a) and use the 'EACH ROW' block to specify that the sum of each row in the range D5:G6 should be calculated (b). It also means that if the average instead of the sum should be calculated, this can be solved by editing only the formula definition in XLBlocks instead of editing a formula and dragging it down. The 'EACH COLUMN' block offers the same functionality for calculations across multiple columns.

5.1.3 Lookups

Excel has five functions that can be used to lookup data (similar to join functions in databases): VLOOKUP, HLOOKUP, INDEX, MATCH, and LOOKUP. They differ in syntax and arguments, and some have problematic defaults [78].

From these formulas, VLOOKUP is used the most frequently, although the combination of INDEX and MATCH is a better alternative. That combination can be used both horizontally and vertically, there are no sorting requirements, and the lookup range can be located anywhere. Unfortunately, it is not frequently used because of it's complexity. It is an excellent candidate to explore if we can replace this with a new function that is easier to use. Therefore we define a general purpose 'LOOKUP' block in XLBlocks (Figure 5.5).



Figuur 5.5: LOOKUP formula in XLBlocks

For this function, the user simply specifies the value they are looking for (a), the range that contains the possible lookup values (b) and the range that contains the matching results (c). XLBlocks translates this into the following formula:

```
=INDEX(C14:C19,MATCH(B5,B14:B19,0))
```

Using the LOOKUP block in XLBlocks give users the flexibility of the INDEX and MATCH combination without the complexity.

5.2 Setup of Think-aloud Study

5.2.1 Research Questions

The goal of this paper is to examine to what degree a block-based formula editor could support professional spreadsheet users while developing or maintaining formulas. To address this goal, we develop XLBlocks, a block-based formula editor for Excel, and conduct a think-aloud study in which participants perform eight typical spreadsheet tasks. After the tasks, we interview the participants and ask them to evaluate the XLBlocks interface, using the CDN framework. For each dimension, we ask them to answer the two following research questions:

- *RQ1:* What are the benefits of XLBlocks regarding this dimension?
- *RQ2:* What are the drawbacks of XLBlocks regarding this dimension?

5.2.2 Participants

We invited thirteen professional Excel users by mail from nine different companies within our network of industrial partners (Table 5.1). Included in the study were participants who use Excel professionally and who use formulas.

Tabel 5.1: Overview of think-aloud study's participants

Nr.	Gender	Age	Functional Domain	Excel level	Frequency	Experience (yrs)
P1	M	38	SE†	9	Daily	15
P2	M	54	SE	7	Daily	20
P3	M	51	SE	8	Daily	10
P4	M	62	Finance	8	Daily	25
P5	M	54	Operations	7	Daily	23
P6	M	25	Operations	7	Daily	4
P7	F	47	Finance	8	Daily	25
P8	M	53	Finance	9	Daily	25
P9	M	39	Finance	7	Daily	20
P10	M	50	CTO	4	Monthly	25
P11	M	56	Finance	8	Daily	25
P12	M	41	SE	8	Daily	20
P13	M	47	Finance	9	Daily	20
Average		47		8		20

† SE = Software Engineering

All participants, except one, use Excel multiple times a day and have on average 20 years of experience using Excel. We asked them to asses their level of expertise with Excel on a ten point scale (one = low, ten = high). We used this scale because it is widely used in (European) schools to grade work and the participants are more familiar with it than for example a Likert scale. On average they rated themselves an eight out of ten.

5.2.3 Think-Aloud Study

In a think-aloud study, we ask the participants to perform eight typical spreadsheet tasks in two different spreadsheets. The tasks are summarized in Table 5.2.

To ensure that the tasks would be similar to the tasks the participants perform in their spreadsheets, we selected formulas that are frequently used in spreadsheets from the Enron corpus [10] and our collection of industry spreadsheets [79].

Spreadsheets are on average used by 13 different users [37], which implies that there are many moments a spreadsheet is transferred from one user to another. Therefore, we

Tabel 5.2: Tasks to be performed by participants in think-aloud study

Task	Description
T1	Create row totals with SUM function
T2	Create column totals with SUM function
T3	Move output of column totals to different range
T4	Lookup account descriptions with LOOKUP function
T5	Create a conditional sum on two conditions with the SUMIFS function
T6	Change SUMIFS formula to calculate sum of a different range
T7	Move output of SUMIFS formula
T8	Explain a formula with a nested IF structure

are interested to see if a block representation of a formula can also support a user in explaining the formula to somebody else, for which, we added task T8.

Before the participants start with the tasks, they receive a ten to fifteen-minute instruction about the XLBlocks interface. They get about 40 minutes to finish the tasks, and we ask them to think-aloud. We provided the participants with two sample spreadsheets and explained each task. The sample spreadsheets are available online[3]. If a participant created a block-model leading to an invalid formula, the formula would not be generated. When such errors occurred during the tasks, we explained the cause of the error.

Five participants performed the tasks individually, the remaining eight participants in pairs. All participants performed tasks T1 to T7, but we asked the participants that worked in pairs additionally to perform T8.

Because of the exploratory nature of the study, we did not measure the time participants needed to complete the tasks. Also, the participants were not quantitatively assessed in performing the tasks. The tasks enabled participants to get experience with XLBlocks so that they could reflect on usability and could provide us with feedback in the interviews.

5.2.4 Interview

After the tasks, we conduct a semistructured interview of about 30 to 40 minutes with each participant. We use the CDN framework to structure the interviews. Participants that perform the tasks in pairs are also interviewed in pairs. We transcribed all interviews, grouped the answers per dimension, complemented it with observations from the think-aloud study, and used the combined information to answer the two research questions per dimension.

The CDN framework has been used in several usability studies [80–83] and Blackwell and Green developed a questionnaire for the CDN [84].

For the interview, we used the dimensions as defined in [85]. We made two adjustments. We excluded the dimension *Abstraction Gradient* because users can not create their own blocks in XLBlocks. Furthermore, we added the dimension *Provisionality* as is described in [26, 86]. The dimension refers to the opportunities that users have to try different design options. Spreadsheets are continuously evaluated, and users are used to how easy it is to try something out. For that reason, we included this dimension. It is clo-

[3]https://doi.org/10.6084/m9.figshare.8863532

sely related to the dimension *Premature commitment*, and we discuss the two dimensions together.

We do not ask participants to fill out the CDN questionnaire, but rather, we use the questions to structure the interviews. It allows us to clarify a dimension if we notice that participants have difficulties understanding it. In addition, we can probe participants for additional details. Finally, for each dimension, we ask participants to grade the usability of both XLBlocks and the built-in Excel formula editor on a scale from 1 to 10.

5.3 Results

This section describes the results of the interviews. An overview is given in Table 5.3 and Figure 5.6. XLBlocks received a better evaluation on all dimensions. The biggest difference is found on the dimension *Secondary notation* and according to the participants, the two interfaces score similarly on the dimension *Hard mental operations*. In the remainder of this section, we present the results of the user-study and interviews per dimension.

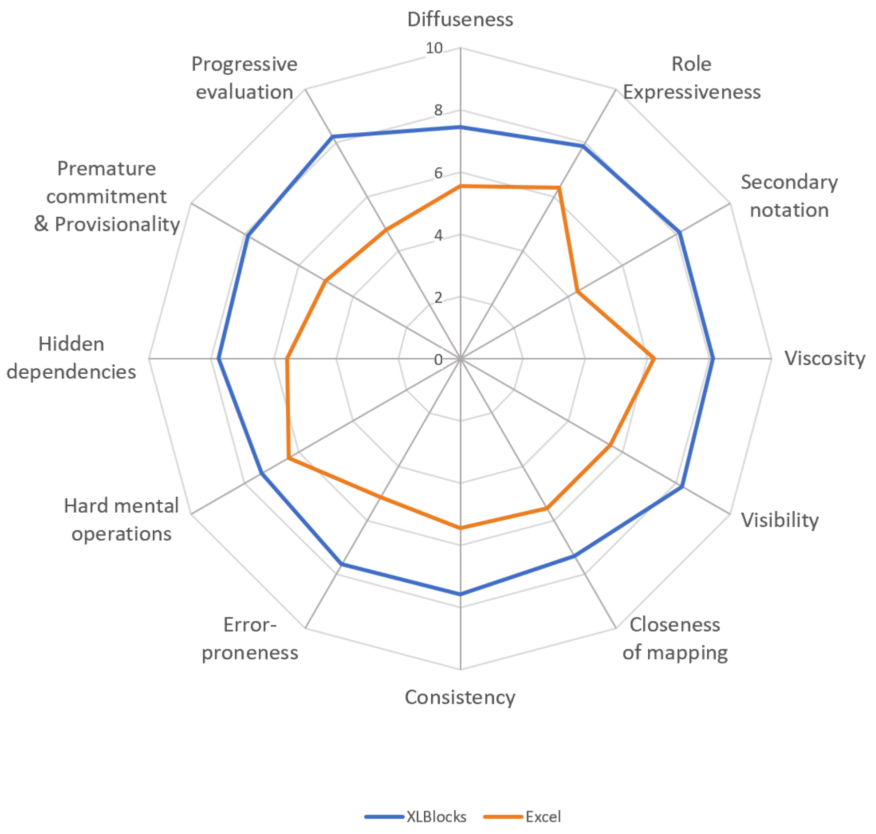
Tabel 5.3: CDN Evaluation of XLBlocks and Excel's formula editor (average score per dimension on a ten point scale)

Dimension	XLBlocks	Excel formula editor
Diffuseness	7.4	5.6
Role-expressiveness	7.9	6.3
Secondary notation	8.1	4.3
Viscosity	8.1	6.2
Visibility	8.2	5.6
Closeness of mapping	7.3	5.6
Consistency	7.6	5.4
Error-proneness	7.6	5.1
Hard mental operations	7.4	6.4
Hidden dependencies	7.8	5.6
Premature commitment & Provisionality	7.9	5.0
Progressive evaluation	8.2	4.8

5.3.1 Diffuseness

The dimension *Diffuseness* describes the verbosity of the language. How many symbols and space is required to express meaning.

RQ1: Benefits XLBlocks: When block-based languages (BBL) are compared to text-based languages, diffuseness is often seen as a drawback for the BBL. They tend to be much more verbose and need more physical space to express the same. However, when we asked our participants about the diffuseness, they considered it as a benefit that more space was available. Excel does not feature a fully equipped integrated development environment (IDE) for editing formulas, but provide the user with a formula bar that is very limited in space (see Figure 1.2b). The additional space the XLBlocs interface provides, makes it easier to read formulas, and because of the graphical nature of the BBL, it is easier to see and recognize the structure of the formula.



Figuur 5.6: Evaluation of formula editor in XLBlocks and Excel on a ten point scale

RQ2: Drawbacks XLBlocks: Some participants (P2, P8, P11, P12) indicate that the XL-Blocks hides parts of the spreadsheet interface, which could be a problem in the case of larger spreadsheets. However, they still prefer the additional space to inspect and understand the formula. Participants P1 and P2 noticed that in the XLBlocks interface when additional functions or arguments are added, formulas tend to grow in width. They would prefer if formulas grow mainly in height.

5.3.2 Role-expressiveness

Role-expressiveness concerns the ease of seeing how a part of a formula relates to the formula as a whole. What is the meaning of every part of the formula?

RQ1: Benefits XLBlocks: According to the participants, the meaning of every individual block is clear. As a reason, they name the text labels on every block. For function names that is obvious, but also the arguments of every function are named (a, b, and c in Figure 5.5). This is not the case in the formula bar, and according to the participants, it makes the reading of the formula much easier. Also the consistent use of colors (all functions have the same color, see also Section 5.3.6: Consistency) support this.

RQ2: Drawbacks XLBlocks: On the other hand, when we observed the participants executing the requested tasks, it became clear that the meaning of the special blocks 'EACH ROW' and 'EACH COLUMN' was not as intuitive as we had hoped. Also the term 'output range' lead to some confusion. Especially, in combination with the 'LOOKUP' function that contained an argument that was named 'Result range' (See also c in Figure 5.5). Also, although constants have a different color than functions, participants P9 and P10 interpreted the constant block that was used to indicate a constant number as the function number (which does not exist in Excel).

5.3.3 Secondary Notation

Secondary notation expresses the options user have to convey additional information about a formula that is not part of the formal syntax of the formula. Spreadsheets as a whole provide ample opportunity for secondary notation. Therefore, it is important to note that we are comparing XLBlocks not with the spreadsheet, but with the formula editor of Excel.

RQ1: Benefits XLBlocks: All participants indicated that it is straightforward to add secondary notation to the formula definition in XLBlocks. XLBlocks has special comment blocks that can be dragged anywhere in the formula. In the excel formula bar, it is not possible to add comments. However, a comment can be assigned to the cell that contains the formula. According to the participants that is not optimal because 1) one can only assign one overall comment for the complete formula, 2) the comment is visible in the spreadsheet model and 3) if a formula has been copied across multiple rows or columns, it is not clear to which formulas the comment belongs.

RQ2: Drawbacks XLBlocks: For secondary notation, the participants, did not mention any specific drawbacks. Participant P2 remarked that further improvement might be possible if a text field was added to the brown formula block (See Figure 5.5) that could be used to write some general information about the formula. On the other hand participants, P1 and P2 asked themselves if they would ever use the comments: *"Because the block definition of the formula is easier to read, the formula documents itself."*

5.3.4 Viscosity

The dimension *viscosity* measures the effort that is required to perform a single change.

RQ1: Benefits XLBlocks: The participants agree that changing a formula is simple in XLBlocks. This was confirmed by the fact that none of the participants made an error in the tasks that involved changing existing formulas. In comparison with the formula bar, it is easy to locate the part of the formula that one wants to change. Also, in the formula bar, users have to be very careful with the use of parentheses, commas, and quotes. This is something that is taken care of automatically in XLBlocks. It was also stated that with XLBlocks it is possible to change a group of formulas with a single edit.

RQ2: Drawbacks XLBlocks: Two participants (P12 and P13) comment that, when editing, they use search and replace a lot. This is possible in the formula bar but not in XLBlocks. Also, when a formula is not very complex, and an edit comprises only changing a few characters, it is much quicker to edit directly in the formula bar than in XLBlocks.

5.3.5 Closeness of Mapping

The dimension *Closeness of mapping* expresses the mapping between the programming world and the problem world.

RQ1: Benefits XLBlocks: All participants agree that XLBlocks visualizes the calculation that will be generated by Excel. Especially, when the formula is complex or nested this is visualized better in XLBlocks than in the formula bar (see Figure 5.7). Also, XLBlocks needs less interpretation. For example, the IF function has three named arguments in XLBlocks: *if*, *then*, and *else*. In Excel, the arguments are not named. It is by convention that the user knows that the first argument is the condition, the second the *then* statement, and the third the *else* statement. Also, the user has to know that commas separate these arguments. In XLblocks, the user does not need to know these conventions.

RQ2: Drawbacks XLBlocks: One participant (P4) argued that the XLBlocks interface has a less direct mapping to the problem the user tries to solve. The EACH ROW and EACH COLUMN blocks add another layer of abstraction, while in the formula bar the user is only editing one cell. The fact that a group of formulas is edited instead of a single formula reduces the closeness of mapping.

5.3.6 Consistency

Consistency means that parts of a formula that have a similar meaning should have a similar appearance.

RQ1: Benefits XLBlocks: Most participants perceived the block notation in XLBlocks as consistent. They name the use of color as the main reason. In XLBlocks each category of blocks has its own color. It is easy to see what is a function, cell reference, or constant. In the formula bar, this is less clear. Although Excel also colors the cell references, there is no difference in typographic style between functions and constants (See Figure 1.2b).

RQ2: Drawbacks XLBlocks: In its current implementation, all functions in XLBlocks have the same color (green). This could result in one big green block of functions if a formula consists of several nested functions. Therefore, one participant (P2) suggested to color the functions according to their category (e.g., text functions would have a different color than math functions).

5.3.7 Error-proneness

The dimension *Error-proneness* indicates how easy it is to make errors while writing a formula.

RQ1: Benefits XLBlocks: In XLBlocks, the code generator inserts commas, parentheses, and quotes at the right place in the formula. According to several participants (P4, P7, P9, P10, and P12), this is the main reason that fewer errors are made in XLBocks. Also, two participants (P3 and P13) observed that the function blocks in XLBlocks guide the user through the function. As a consequence, it is not possible that arguments are forgotten or used in the wrong order.

RQ2: Drawbacks XLBlocks: During the tasks, we observed participants placing blocks in the wrong position. This corresponds with feedback we received from other participants (P9, P10, and P12) that it was possible to misplace blocks. Also, a participant forgot to give the formula a name. Although this does not lead to an error in the formula, it makes it more challenging to find the formula.

5.3.8 Hard Mental Operations

Working on a formula requires mental effort. The dimension *Hard mental operations* indicates to which extent the block-based language used in XLBlocks itself causes this mental effort.

RQ1: Benefits XLBlocks: The participants stated that in XLBlocks they had to think less about the exact syntax of the formula, the correct order of the arguments and when nesting functions, the order of the functions themselves.

RQ2: Drawbacks XLBlocks: For the participants, the concept of the 'EACH ROW' and 'EACH COLUMN' blocks was the most difficult to understand. Partially this is not something caused by the notation, but by the concept itself. On the other hand, participants had difficulties placing the 'EACH ROW' block at the right spot. They could use more guidance from the XLBlocks interface.

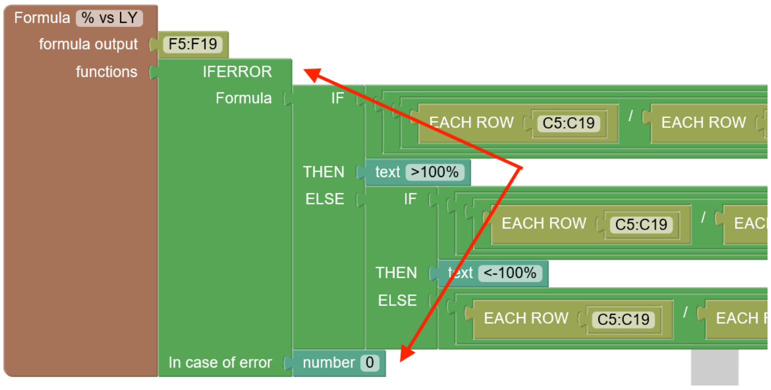
5.3.9 Hidden Dependencies

Spreadsheets are disreputable for their hidden dependencies. However, they are less common in a spreadsheet formula. A hidden dependency in a spreadsheet could be a function and its corresponding arguments. Consider, for example, the following formula:

```
=IFERROR(IF(C5/D5-1>1,">100\%",IF(\\C5/D5-1<-1,"<-100\%",C5/D5-1)),0)
```

The zero at the end of the formula is an argument of the IFERROR function at the beginning of the formula and could be considered as a hidden dependency.

RQ1: Benefits XLBlocks: According to the participants, it is much easier to see hidden dependencies in a formula in XLBlocks than in the formula bar in Excel. They name the previously mentioned IFERROR function as an example (Figure 5.7). Because the 'in case of error' argument is physically connected to the IFERROR function, the dependency is much more visible.



Figuur 5.7: Example of the IFERROR function, the 'in case of error' argument is visually connected to the function

RQ2: Drawbacks XLBlocks: The participants did not mention any drawbacks of XLBlocks concerning *Hidden dependencies.*

5.3.10 Premature Commitment & Provisionality

The dimension *Premature commitment* expresses the extent to which the user is forced to take decisions before the necessary information is available. This dimension was first defined in [85]. Later it was extended in [86] and [26] with a separate dimension *Provisionality* that concerns the opportunities for the user to play around with ideas. Is it, for example, easy to try different design options or to use ‘what-if’ scenarios?

RQ1: Benefits XLBlocks: According to the participants, the user has complete freedom in the order that blocks are dragged onto the canvas. During the tasks, we observed that dragging out the necessary blocks helped the participants to think about the formula. They said things like *“I will start with the blocks I know ...”* (P8) and *“Now I start seeing how it will come together ...”* (P7). They compared it with the formula bar where users are forced to start with the outer function and work their way inwards. Participants also recognized that it was possible to create two variants of a part of the formula and that they could easily swap them to try out the different options. They liked that it was not necessary to ‘clean up your canvas’.

RQ2: Drawbacks XLBlocks: The participants did not mention any drawbacks of XLBlocks concerning *Premature commitment* and *Provisionality*.

5.3.11 Progressive Evaluation

While developing a formula, users often want to check if the formula gives the desired results. The dimension *Progressive evaluation* focuses on how a notation facilitates this.

RQ1: Benefits XLBlocks: The participants perceived the way XLBlocks makes it possible to check the formula during development, more comfortable to work with than the Excel formula bar. As main reason, they named the blocking error they get in Excel when the formula is not correct.

RQ2: Drawbacks XLBlocks: One participant would prefer if XLBlocks showed a real-time version of the formula that is being generated.

5.4 Discussion

5.4.1 Learnability

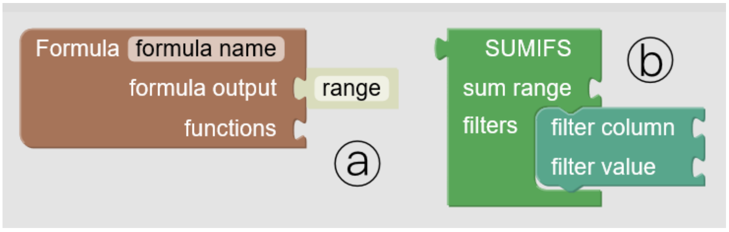
We know that block-based languages perform well on learnability [87], [88]. This was confirmed in our think-aloud study. After receiving only 15 minutes of instruction, all participants were able to finish the tasks. Several participants (P1, P2, P7, and P8) remarked that it surprised them how easy it was to learn to work with XLBlocks.

We also observed that the participants were able to create a formula with the *lookup* function in XLBlocks, which does not exist in Excel. It shows that even if the function is unknown, the blocks are intuitive enough that users can work with it.

5.4.2 Further Reduce Error-Proneness

According to the participants, it is less likely to make errors in XLBlocks than in the formula bar of Excel. Nevertheless, based on the feedback we received during the think-aloud study, it can be further improved. In the current version of XLBlocks, it is allowed to connect invalid combinations of blocks (e.g., connecting a function block where a cell reference block is expected). This can be solved if more robust type-checking is implemented in XLBlocks.

Users could be further supported by a preconfigured group of blocks. Instead of dragging a formula block and a range block from the toolbox and connect them on the canvas, they can drag a preconfigured formula block with the output range already attached. Blockly offers this feature with shadow blocks (placeholder blocks) or block groups (Figure 5.8) [89].



Figuur 5.8: An example of a formula block with a *shadow* range block (a) and a block group consisting of A SUMIFS block and a filter block (b).

Inspired by this concept, some participants suggested the idea to create your own templates of blocks for formula patterns that you use frequently and store them on your toolbox (similar to the backpack feature in Scratch [90]).

5.4.3 Simultaneous Use of XLBlocks and the Formula Bar

XLBlocks has been developed as an alternative way to edit a formula in Excel. It is not meant to replace the formula editor of Excel but to complement it. XLBlocks present the block-based formula embedded into the spreadsheet itself. Some participants (P10 and P13) named this as one of the strengths of XLBlocks. According to them some tasks, like making small edits in simple formulas, are better suited for the formula bar, while other tasks like editing complex formulas or explaining a formula to somebody else are better suited for XLBlocks.

With XLBlocks it is possible to develop a block-based specification of a formula and generate a valid spreadsheet formula. The other direction: generating a block-based formula from a spreadsheet formula, is not yet implemented. Several participants would like to see this feature added. Especially when explaining complex formulas to others, it would be convenient if the user could click on the formula and see the block-based representation in XLBlocks and use that as a basis for the explanation.

5.4.4 Threats to Validity

A threat to the external validity of our evaluation concerns the representativeness of the participants. Future studies are necessary to generalize our findings.

Another threat to the external validity of our evaluation is the representativeness of the tasks the participants had to perform. To mitigate this, we used formulas that professional spreadsheet users use in real-life spreadsheets.

We did not randomly select our participants, which is a threat to the internal validity. Nevertheless, we believe the group of participants serve as a useful reference group since they all work with spreadsheets daily and have on average 20 years of experience using Excel. Furthermore, they work at different companies, have different backgrounds, and work in different functional domains.

We are both the designers of XLBlocks and the interviewers, and this is another internal threat to the validity of our evaluation. We minimized this threat to use the CDN framework to guide the questions in the interview and making sure that all aspects of the usability of the XLBlocks interface were covered.

5.5 Related Work

5.5.1 Spreadsheets and Visual Languages

Most related to our research is the work of Burnet *et al.* [91], Leitão and Roast [92], Sarkar *et al.* [93], and Abraham *et al.* [83].

Burnett *et al.* developed Forms/3, a general purpose visual language that builds upon the spreadsheet paradigm. They leveraged the visual aspect of spreadsheets and at the same time tried to overcome some of the spreadsheet limitations like a limited number of types and the lack of abstraction capabilities. However, they consider the spreadsheet as a whole as a visual language, while in this paper, we focus on a visual language for the formula editor.

Leitão and Roast designed a visual language to represent spreadsheet formulas graphically. They developed two variants: an 'Explicit Visualization' (EV) and a 'Dataflow Visualization' (DV). In both variants, numeric values, cell references, strings, operators and built-in spreadsheet functions are represented in different combinations of shape and color. In EV, the visualized formula is a visual match of the original textual formula, while in DV the formula is presented hierarchically in a syntax tree. Both are inspired by dataflow diagrams and are less textual than a block-based language.

Sarkar *et al.* propose multiple-representation editing in spreadsheets. They introduce Calculation View, an alternative representation of the spreadsheet, primarily designed for viewing formulas and their groupings. They use a new textual syntax for copying a formula into a block of cells and naming cells or ranges. Calculation View uses a textual notation in a columnar grid of pseudocells to maintain similarity with the spreadsheet grid.

Abraham *et al.* introduced ViTSL, a visual specification language for spreadsheets. The language allows the definition of spreadsheet templates that can be used by a spreadsheet generator to create Microsoft Excel spreadsheets automatically [19]. Derived from ViTSL, Engels and Erwig developed ClassSheets [94]. A ClassSheet represents both the structure and relationships of the involved (business) objects within the spreadsheet. It narrows the semantic distance between a problem domain and a spreadsheet application. A drawback of this approach was the lack of connection between the stand-alone model development environment where the ClassSheet was defined and the spreadsheet itself. As a result, automatic synchronization between the model and the spreadsheet was not possible. Cunha *et al.* [95] embedded the ClassSheet in the spreadsheet itself and made co-evolution of the model and the spreadsheet possible.

5.5.2 Cognitive Dimensions of Notation

Green and Petre have used the CDN framework as an evaluation technique for visual programming environments [85]. It provides a vocabulary for discussing the usability of programming languages. In their study, they present an outline of the cognitive dimensions and use them to evaluate two different visual programming environments.

The CDN were also used to design questionnaires intended for users to evaluate the usability of programming tools. In reaction, Blackwell and Green developed a generalized questionnaire and conducted a pilot study that used the questionnaire with a wide range of respondents [84]. The results of that study showed that the CDN questionnaire is a suitable tool for user evaluation of programming languages, tools, and environments.

The framework has been used to evaluate multiple programming languages. Most related to our research is the previously mentioned research of Abraham *et al.*. They used the CDN to evaluate their visual specification language ViTSL.

5.5.3 Block-Based Languages

Glinert introduced in 1986 the language BLOX, which can be considered as the first block-based language [96]. Research into block-based languages increased after the introduction of languages like Alice [97], Scratch [75], and Blockly [77]. These languages were designed as programming environments for younger learners.

Related to our research is the work of Weintrop *et al.* [98]. They created CoBlox, a block-based interface for programming a one-armed industrial robot. They showed that block-based programming could make a complex task like programming an industrial robot accessible for adults with limited programming experience.

Also related to our research is the study of Holwerda and Hermans [99]. They conducted a user study with Ardublockly, a block-based language derived from Blockly. In their study, they focus on gaining an understanding of the strengths and weaknesses of block-based languages as seen by professionals.

5.6 Concluding Remarks

This paper aims to explore if editing spreadsheet formulas can be eased by using a block-based formula editor. We, therefore, developed XLBlocks, a block-based formula editor, and conducted a think-aloud study in which we evaluated the usability of the editor using the CDN framework. XLBlocks received on all dimensions a better evaluation. The difference was the most notable for the dimensions *Secondary notation*, *Error-proneness*, *Progressive evaluation*, *Premature commitment* and *Provisionality.*

Users recognized that in XLBlocks, they do not have to consider the correct syntax of functions. Also editing a part of a formula is easier in XLBlocks because they can easily drag and drop different parts of the formula on the canvas. During the think-aloud study, we observed that dragging different blocks on the canvas also supported the user in thinking about the formula. They started with the blocks they were sure about and then focused on the more difficult parts. They appreciated that they could freely decide in which order they would build the formula.

We also received feedback on areas that could be further improved. Varying the colors of the function block (e.g., by category) could make the formulas easier to read. In the current prototype, it is possible to connect invalid combinations of blocks. This could be solved by more robust type-checking and provide block groups as templates.

This paper gives rise to several directions for future work. At the moment, XLBlocks can generate a valid Excel formula from a block-based specification. We will extend the prototype with the possibility to generate a block-based representation of a spreadsheet formula. Furthermore, we are planning to make XLBlocks aware of the structural changes a user makes to the spreadsheet like inserting or deleting rows and columns. Finally, we will explore the possibility to facilitate the user to create templates for formulas that they use frequently.

6
The Effect of a Block-based Language on Formula Comprehension in Spreadsheets

The use of spreadsheets in industry is widespread. It is known that spreadsheets have an average life span of five years, and during this life span, they are used on average by thirteen different persons. Consequently, spreadsheets need maintenance, and knowledge about the spreadsheet needs to be transferred from one user to another. To minimize the risk of introducing new errors, a thorough understanding of the spreadsheet's formulas is needed during maintenance and knowledge transfer tasks.

Research on the use of block-based languages has shown that they positively affect the comprehension of program code. We hypothesize that using a block-based representation of a spreadsheet formula will positively affect formula comprehension.

Hence, we extended XLBlocks, a block-based formula editor for spreadsheets, with the functionality to generate a block-based representation of an existing formula. We conduct a think-aloud study with twenty-one experienced spreadsheet users from industry and ask them to perform a set of spreadsheet comprehension tasks using XLBlocks. During an interview, we ask them, using the Cognitive Dimensions of Notations framework, to reflect on the use of XLBlocks.

We found that participants preferred to use the block-based representation of formulas when analyzing or explaining formulas or to implement non-trivial changes. Named function parameters and the absence of parentheses and commas make functions easier to understand. Furthermore, the visualization enables the user to separate smaller parts in the formula, which improves comprehension. Finally, the possibility to navigate from formula to formula makes it clear how formulas work together and improve the understanding of the spreadsheet as a whole.

This chapter has been published as *B. Jansen and F. Hermans. The Effect of a Block-based Language on Formula Comprehension in Spreadsheets, ICPC21* [34]

Spreadsheets are ubiquitous in industry and often used for critical business decisions. Unfortunately, spreadsheets are also known for their error-proneness. Almost all spreadsheets contain non-trivial errors[11]. Consequently, companies are at risk of basing their decisions on inaccurate information, which can lead to significant loss of money or reputation.[1]

A major part of spreadsheet research is focused on improving spreadsheets by applying software engineering methods. For example, the concept of testing in spreadsheets was studied by Rothermel *et. al.* [44] and more recently by Roy *et. al.* [20]. Hermans *et. al.* [45] and Cunha *et. al* [46] introduced the idea of reverse engineering of spreadsheets and designed methods for extracting class diagrams from spreadsheets. Several studies [57] [38] [58] define and investigate code smells in spreadsheets. Refactoring is closely related to code smells, and both Badame and Dig [16] and Hermans and Dig [15] developed tools for refactoring in spreadsheets.

The common denominator in these studies is that they provide methods and techniques that support users in improving spreadsheets. Nevertheless, a focus on spreadsheet comprehension is lacking. According to Hermans *et. al.* [6], spreadsheets have an average life span of five years and are on average used by thirteen different users. This means that during a spreadsheet's lifetime, maintenance is needed, and for that, knowledge needs to be transferred from one user to another. During these 'transfer scenarios', a thorough understanding of the spreadsheet minimizes the risk of introducing new errors.

Therefore, we focus in this paper on formula comprehension. In an earlier study [32] we introduced XLBlocks, a block-based formula editor for spreadsheets. With this editor, it is possible to create formulas with a block-based language instead of the default textual formula language and translate them automatically into valid spreadsheet formulas. However, in our first implementation of XLBlocks, it was impossible to generate a block-based representation from a formula, making it less suitable for formula comprehension. For this study, we have extended XLBlocks with the functionality to generate from a textual formula a block-based representation of that formula. This enables users to analyze existing spreadsheet formulas in a block-based language.

In this paper, we want to understand the effect of a block-based language for spreadsheets on formula comprehension. To answer this question, we conduct a think-aloud study in which we ask participants to perform a set of formula comprehension tasks with a new version of XLBlocks. When they have completed these tasks, we interview them and ask them to reflect on their experience with XLBlocks. To guide the interview, we use the Cognitive Dimensions of Notation (CDN) Framework [26].

6.1 Related Work

6.1.1 Spreadsheets and Visual Languages

Related to our study is the work of Burnet *et. al.* [91]. They introduced the visual research language Forms/3. This language's goal was to eliminate some of the spreadsheet systems' limitations without abandoning the spreadsheet paradigm. The language describes a com-

[1]http://www.eusprig.org/horror-stories.htm

plete spreadsheet model. This in contrast to XLBlocks, where we focus on an individual formula.

Abraham *et. al.* [83] also introduced a visual language for spreadsheets called ViTSL. With ViTSL, it is possible to define a spreadsheet template. From such a template, a spreadsheet can be generated automatically. Based on this work, Engels and Erwig [94] introduced ClassSheet. A ClassSheet represents both the structure and the relation between (business) objects within the spreadsheet. With ClassSheets, the problem domain and spreadsheet domain are brought closer together. The ClassSheets needed to be developed in a stand-alone application, and for that reason, real-time synchronization between the ClassSheet and the spreadsheet was not possible. Cunha *et. al.* [95] integrated ClassSheets in the spreadsheet and enabled two-way synchronization between the ClassSheet and the spreadsheet.

Leitão and Roast [92] developed a visual language for formulas. It is not a block-based language but a data-flow language. They worked on two different variants: Explicit Visualization (EV) and Data flow Visualization (DV). In the EV, the visualization is a direct match of the spreadsheet formula. Operators, cell references, constants, and functions have been replaced by symbols. The DV uses the same symbols, but they are presented as a syntax tree.

Finally, Sarkar et al. [93] introduced Calculation View, which is also an alternative representation of the spreadsheet. Formulas are presented in a textual calculation view adjacent to the standard grid view. One of Calculation View features is 'range assignments', which allows the user to assign the same formula to a range of cells. This is more efficient and less maintenance intensive than manually dragging the formula down to a range of cells. Furthermore, with Calculation View, it is possible to name cells easily and refer to those names in other formulas.

6.1.2 Block-based languages

BLOX can be considered the first block-based language and was introduced by Glinert [96]. After the introduction of several block-based languages, like Alice [97], Scratch[75], and Blockly [77] the body of research on block-based languages started to grow.

Most related to our research is a study of Weintrop *et. al.* [98]. In this study, they introduce CoBlox, a block-based language to program a one-armed industrial robot. They demonstrate that block-based languages are not only suitable for children in an educational environment but also useful for adult novice programmers in an industrial setting. Adult programmers successfully implemented robot programs with CoBlox faster and with no loss in accuracy than similar programmers using one of two widely-used industrial robot programming approaches. They also scored better on usability, learnability, and overall satisfaction.

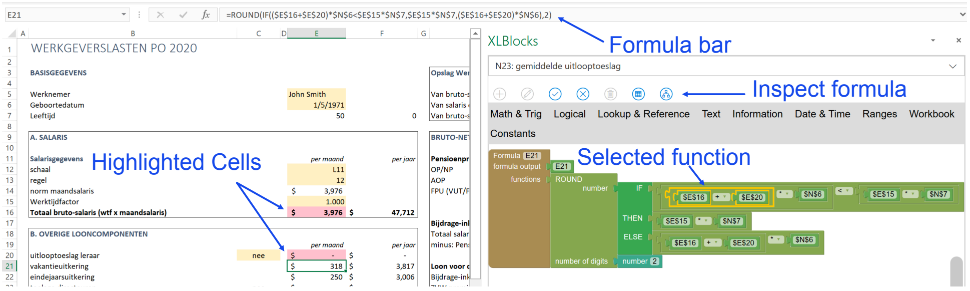
Weintrop *et. al.* also conducted a study on block-based comprehension [33]. They asked participants to answer twenty program comprehension questions using two variants (text-based and block-based) of pseudocode developed for the Advanced Placement CS Principles course. They concluded that learners performed better on questions presented in the block-based modality.

Finally, Hermans and Holwerda [100] conducted a user study with ArduBlockly. Using a user study, they demonstrate a usability analysis of block-based editors based on the Cognitive Dimensions of Notation (CDN) framework. Furthermore, they give an overview of several design maneuvers to improve programming time and effort, program comprehension, and programmer comfort.

6.2 Analyzing Formulas with XLBlocks

6.2.1 XLBlocks Interface

Figure 6.1 shows the XLBlocks interface. The spreadsheet is displayed on the left side of the screen, the XLBlocks editor on the right side. At the top of the screen is the formula bar, which is the Excel default tool to enter formulas. A formula can be analyzed in XLBlocks by selecting the cell with the formula and using the inspect formula button to generate the formula's block-based representation. A video demonstrating the user interface of XLBlocks is available on-line.[2]


E21 =ROUND(IF(($E$16+$E$20)*$N$6<$E$15*$N$7,$E$15*$N$7,($E$16+$E$20)*$N$6),2)

Figuur 6.1: Left the traditional view with the formula bar at the top, right the XLBlocks interface showing the block representation of the formula

6.2.2 Generate Block-based Formulas

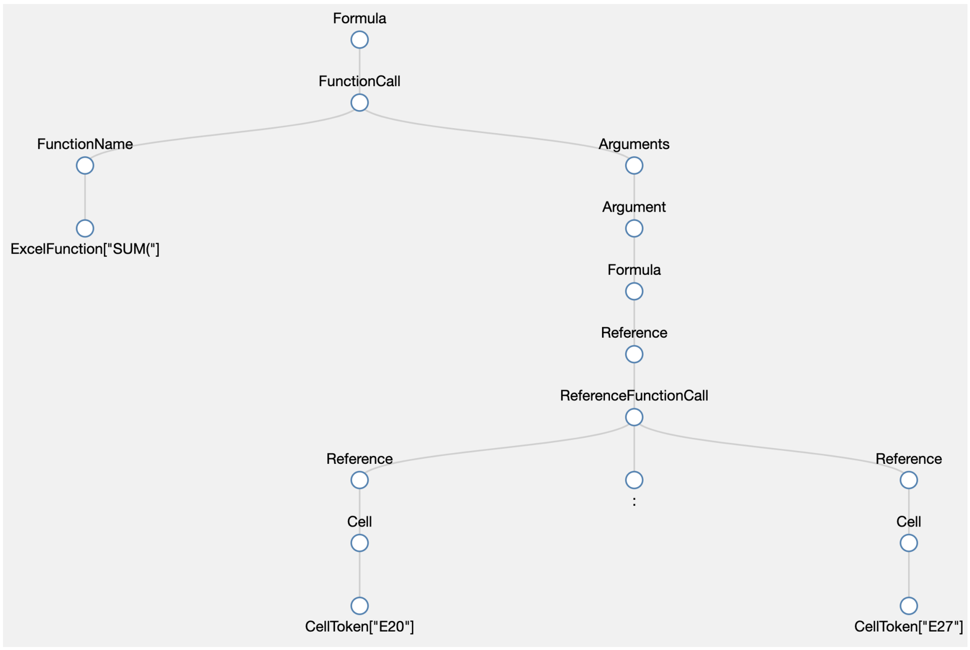
We extended XLBlocks with the functionality to generate a block-model of a formula from the textual formula. To do so, we use XLParser, a parser developed by Aivaloglou *et. al* [79] [101] that produces a parse tree for spreadsheet formulas. In XLBlocks, this parse tree is converted into an XML definition of the block model, which is then translated to the formula's block-based representation (see Figure 6.2).

6.2.3 Highlighting and scrolling

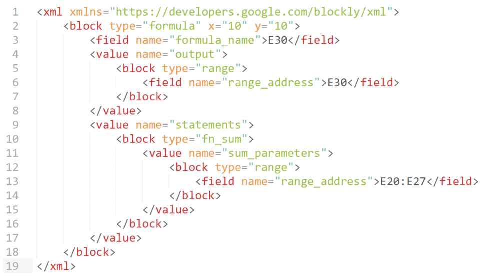
Spreadsheet formulas often use the outcome of other formulas in their calculation. To completely understand a formula, users need to trace the formula's precedents. To support users in understanding the formula and tracing its precedents, XLBlocks highlights cells in the spreadsheet that are referred to in the formula. In Figure 6.1 a part of the formula in XLBlocks is selected (highlighted by a yellow border). The cells referred to in this selected part of the formula are also highlighted in the spreadsheet.

Some cells might be out of the user's field of view. To inspect these cells, a user can select a single range block, and XLBlocks will automatically scroll the cursor to that cell

[2]https://doi.org/10.6084/m9.figshare.14268017.v1

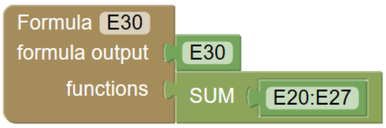


(a) Syntax tree



```
<xml xmlns="https://developers.google.com/blockly/xml">
    <block type="formula" x="10" y="10">
        <field name="formula_name">E30</field>
        <value name="output">
            <block type="range">
                <field name="range_address">E30</field>
            </block>
        </value>
        <value name="statements">
            <block type="fn_sum">
                <value name="sum_parameters">
                    <block type="range">
                        <field name="range_address">E20:E27</field>
                    </block>
                </value>
            </block>
        </value>
    </block>
</xml>
```

(b) XML Definition of block



(c) Block representation of SUM function

Figuur 6.2: Generation of block-based formula and make it the active cell. If needed, the user can immediately inspect that formula and navigate from precedent to precedent to analyze the complete calculation chain. For comprehension, the user must see the cell, its content, and, preferable, the corresponding label. In most spreadsheet models, the label can be found to the left or above the cell. Therefore, XLBlocks ensures that the columns to the left and the rows above the cell are visible when a user scrolls to a cell.

6.2.4 Implementation

XLBlocks has been implemented as an Excel Add-in. It has been developed with the Excel JavaScript API [76]. The Blockly Library [102] [77] is used to develop the visual programming editor and was extended with custom blocks and code generators for the definition and generation of spreadsheet formulas. Twenty-tree different spreadsheet functions have been implemented in the research prototype of XLBlocks. Among these twenty-three functions are the fifteen most frequently used functions of the Enron Corpus [10]. The current research prototype of XLBlocks can only analyze formulas on the same worksheet.

6.2.5 Research Questions

Spreadsheets have a long lifespan, and thus spreadsheets need maintenance. Furthermore, there will be several transfer moments during their lifespan where knowledge about the spreadsheet needs to be exchanged between different users. For both maintenance tasks and the transfer scenarios, a thorough understanding of the formulas is essential.

Therefore we focus in this study on the effect of a block-based formula language on formula comprehension in spreadsheets. In this paper, we will answer the following research questions:

- *RQ1:* What is the effect of a block-based formula editor on the users' ability to understand a formula?
- *RQ2:* What is the effect of a block-based formula editor on the users' ability to explain a formula to somebody else?
- *RQ3:* What is the effect of a block-based formula editor on the users' ability to understand the spreadsheet model as a whole?

6.2.6 Participants

We invited forty-three professional Excel users by e-mail from twenty-eight different companies. We were looking for experienced Excel users that use Excel in their professional lives. Twenty-one of them, working for thirteen companies, accepted the invitation (see Table 6.1). Five of them had participated in our earlier study [32].

All participants use Excel professionally, are accustomed to working with formulas, and have on average twenty-five years of experience with Excel. Most of them use Excel daily. We asked them to assess their level of expertise with Excel on a ten-point scale (one = low, ten is high). This form of rating is widely used in (European) schools, and the participants are more familiar with it than, for example, a five-point Likert scale. On average, they rated themselves an eight out of ten.

Tabel 6.1: Overview of think-aloud study's participants

Nr.	Gender	Age	Functional Domain	L[a]	F[b]	E[c]
P1	M	52	Engineering	7	D	35
P2	F	48	Controlling	8	D	25
P3	M	38	Controlling	8	D	25
P4	M	44	IT	8	D	20
P5	M	59	Finance & Control	7	D	35
P6	M	38	Consultancy	9	D	25
P7	M	43	Finance	8	D	20
P8	M	60	Finance	6	D	30
P9	M	38	Finance	8	D	18
P10	M	64	Finance	8	D	30
P11	M	55	Data analytics	8	W	30
P12	F	46	Business Intelligence	8	D	20
P13	M	49	Finance & Control	7	D	25
P14	M	49	Consultancy	6	D	25
P15	F	60	Consultancy	7	D	25
P16	M	55	General Management	7	M	25
P17	M	45	Business Control	7	D	20
P18	M	44	Finance & Control	8	D	24
P19	M	48	Finance	8	D	25
P20	M	38	Finance	9	D	20
P21	M	55	Finance	8	D	29
Average		49		8		25

[a]Excel level, [b]Frequency: (D)aily, (W)eekly, (M)onthly, [c]Experience (yrs)

Tabel 6.2: Pacione's comprehension activities

Activity	Description
A1	Investigating the functionality of the system
A2	Adding to or changing the system's functionality
A3	Investigating the internal structure of an artifact
A4	Investigating dependencies between artifacts
A5	Investigating run-time interactions in the system
A6	Investigating how much an artifact is used
A7	Investigating patterns in the system's design
A8	Assessing the quality of the system's design
A9	Understanding the domain of the system

6.2.7 Comprehension Tasks and Think-Aloud Study

In a think-aloud study, we asked participants to perform twelve comprehension tasks on an existing spreadsheet model. We used the framework designed by Pacione *et. al* [52], commonly used for empirical evaluation of code comprehension [53] [54] and used their set of nine comprehension activities (see Table 6.2)

In previous research [30] we have translated Pacione's software comprehension tasks to the spreadsheet domain. In this study, we use similar tasks. Each tasks covers at least one of the activities in Table 6.2 and all tasks combined to cover all activities (see Table 6.3).

Tabel 6.3: Overview of comprehension tasks

		Comprehension Activities							
Nr.	Task	A1	A2	A3	A4	A6	A8	A9	Total
T01	Explain a calculation	X		X				X	3
T02	Adapt a calculation		X	X					2
T03	Explain a key concept of the model	X		X				X	3
T04	Find and correct an error		X	X					2
T05	Correct an error		X	X					2
T06	Determine relationships between two cells				X	X			2
T07	Find dependents of a cell				X	X			2
T08	Explain how the spreadsheet can be improved			X			X		2
T09	Assess adaptability of the spreadsheet						X		1
T10	Assess transferability of the spreadsheet						X		1
T11	Explain a calculation	X		X				X	3
T12	Explain a calculation	X		X				X	3
Total		4	3	8	2	2	3	4	26

Because of the liveness of a spreadsheet, there is no clear distinction between coding and runtime. We, therefore, excluded activities A5 and A7.

The spreadsheet we use for the study is defined by the Dutch Primary Education Board. Schools can use it to calculate the annual salary costs of their employees. We choose this model because it is publicly available[3], and contains twelve of the fifteen most frequently used functions in the Enron Corpus [10]. We adapted the model to incorporate the three missing functions and moved all lookup tables to the same sheet as the calculation model[4].

6.2.8 Think-Aloud study

We organized on-line meetings with the twenty-one participants. We used either Microsoft Teams or GoToMeeting to facilitate these meetings. At the start of every meeting, we checked the participants' monitor and resolution, shared our screen with them, and asked if the different interface elements were readable. For one participant, we changed the zoom factor of the XLBlocks interface from 80% to 110%.

[3]https://www.poraad.nl/files/themas/financien/werkgeverslasten_po_2020.xlsx
[4]Adapted model available at: https://doi.org/10.6084/m9.figshare.14268017.v1

Before the meeting, we sent each participant an instruction video about XLBlocks, asking them to watch it before participating in the study. Two participants were not able to do this. At the start of their meeting, we provided the instruction live, using the same script that was used for the video.

We shared our screen with the participants during the meeting and gave them control over our keyboard and mouse. We also recorded the meeting on video. We asked the participants to perform the twelve comprehension tasks and to think-aloud during the study. If they felt silent while performing the tasks, we gave them a quick reminder to express their thoughts.

6.2.9 Interview

Immediately after the comprehension tasks, we conducted a 45-minute interview. We used the Cognitive Dimensions of Notation (CDN) framework to structure the interview. The CDN Framework has been used in several usability studies [80–83, 99], and Blackwell and Green developed a questionnaire for it [84]. In our interview, we covered the dimensions as defined in [26]. We excluded the dimension *Abstract Gradient* because, in XLBlocks, users can not create their own blocks.

We did not ask the participants to fill out the CDN questionnaire, but rather, we used the questions to structure our interview[5]. It allowed us to clarify a question, and it enabled us to probe participants for additional details.

We grouped all participants' answers per cognitive dimension and complemented them with our observations and the participant's remarks from the think-aloud study. We used the combined information to answer our research questions. These findings will be presented in the next section.

6.3 Results

In this section, we present the results of the think-aloud study. First, we will cover the execution of the different comprehension tasks, after that the findings from the CDN interview, and we end this section by answering the research questions.

6.3.1 Comprehension Tasks

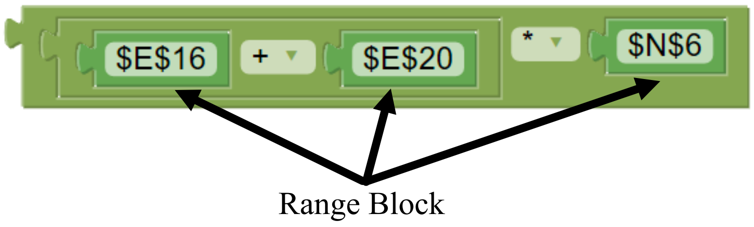
All twenty-one participants were able to perform ten of the twelve comprehension tasks. In the next paragraphs, we will describe our findings in detail. We grouped the findings by the comprehension activities as defined by Pacione *et al.*[52].

Investigating the functionality of (a part of) the system

By design, XLBlocks displays a single formula. Nevertheless, several features helped the participants to get an understanding of the spreadsheet model as a whole.

In the block-based representation of the formula, each cell reference is represented by a range block (see Figure 6.3). If a user clicks on any block in the formula, XLBlocks will highlight all cells in the spreadsheet with a range block in that part of the formula. This gives the user an overview of the other cells in the spreadsheet that are involved in the calculation of this formula. If users click on a single range block, they can open that

[5]Interview questions available at:https://doi.org/10.6084/m9.figshare.14268017.v1



Figuur 6.3: Example of range blocks in XLBlocks

formula in XLBlocks to analyze it. In this way, users are supported in understanding the working of the spreadsheet model as a whole.

In the current implementation, XLBlocks highlights all cells in the same color. Some participants suggested that it would be helpful if the highlighted cells had a unique color and that the selected range blocks would light up in the same color.

Adding or changing the system's functionality

We confronted the participants with two erroneous formulas, and all participants were able to find and correct the errors in these formulas. Furthermore, they were able to make a change in a complex formula. They had to replace a nested IF structure with a VLOOKUP function. According to the participants, this was easier to perform in XLBlocks than in the formula bar. In XLBlocks, it is possible to drag the complete IF structure out of the formula and replace it with a VLOOKUP function block. They did not need to consider the exact placement of parentheses and commas', making the change more straightforward and quicker.

One of the errors the participants needed to find was a SUM function in which some cells were omitted. The highlighting of the involved cells in the spreadsheet helped to visualize the mistake, but Excel offers the same functionality in the formula bar. Eight participants remarked that for such a small change (extending the range of a SUM function), the formula bar is more efficient than XLBlocks.

The second error that participants had to correct was a logic error in an IF formula. In this case, according to the participants, it was easier to spot the error in XLBlocks because the formula as a whole was easier to read.

Investigate the internal structure of an artifact

We asked the participants to explain several formulas with XLBlocks. All participants were able to do that, even if they had no domain knowledge of the spreadsheet model. In XLBlocks, in contrast to the formula bar, the parameters of a function are named. According to the participants, that makes it easier to understand a formula. For example, P19 stated: *"In XLBlocks, the IF, THEN, and ELSE parts of the formula are visible in the IF Block. That is not the case in the Excel formula bar."*

When participants explained a formula, we observed that it took significant time to look up a cell reference in the spreadsheet to determine its meaning. If the same reference is used more than once in the formula, it was not uncommon that participants had to look up the meaning again. Some participants suggested that formulas would be easier to read if the cell references had meaningful names instead of the abstract A1 naming style that is the default in Excel (see also Section 6.4.2).

Several participants noted that in XLBlocks, a formula is visually split into different components (instead of a long string of characters in the formula bar), making it easier to understand the formula.

Investigating dependencies between artifacts and how much an artifact is used
The participants were asked to trace the precedents of five different formulas. They were all able to do that. XLBlocks provide two features that supported them in these tasks:

1. When the formula is selected in XLBlocks, all cell references used in the formula are highlighted in the spreadsheet.

2. If a user selects a single cell reference in XLBlocks, they can immediately inspect the formula in that cell and quickly navigate from formula to formula.

Assessing the quality of the system's design
We asked the participants how they would improve the spreadsheet model we used during the comprehension tasks. Two of the twenty-one participants were not able to come up with some improvements. The other participants mentioned improvements like:

- separating input, calculation, and output of the model to different worksheets

- move all parameters and lookup tables to different worksheets

- in the current model, some calculation are based on monthly amounts, other on yearly amounts. Several participants proposed to make these calculations consistent. Either all based on monthly or on yearly amounts, but not mixing them.

Understanding the domain of the system
After working with the spreadsheet model in XLBlocks, participants had a better understanding of the functional domain (salary administration). One of the questions was if they could tell which components make up the total salary costs. Furthermore, they had to explain the calculation of several pension premiums. To calculate this, you have to take into account an exemption amount. If your income is lower than the exemption amount, you do not have to pay a premium; however, if your income is higher than the exemption amount, you pay a percentage of your income minus the exemption amount. When the participants had to explain these types of formulas, they first read the formula aloud, and next, they would summarize the logic of the formula in their own words, recognizing the pattern described above.

6.3.2 CDN Interview

In the next paragraphs, we will present the results of the interview. We will group the observations by the different Cognitive Dimensions [26].

Viscosity
The dimension *Viscosisty* expresses the resistance to change in a language and consequently has more impact on maintenance than comprehension. We included it in the interview because one of the comprehension activities (A2 in Table 6.2) involves adding or changing the system's functionality. XLBlocks has a drag and drop interface. The mouse or trackpad is the primary input device. In general, this slowed the user down, which several participants confirmed. Because of this, participants would prefer to make small changes in the formula bar instead of XLBlocks.

For complex formulas, this was different. In one of the tasks, they had to replace a nested IF structure with a VLOOKUP function. This is not easy in the formula bar and involves careful placement of the cursor, ensuring that one is not selecting one parenthesis too many or too few. Implementing this change goes faster in XLBlocks. The user does not have to bother about parentheses and can easily drag the IF structure out of the formula and replace it with a VLOOKUP block.

Finally, some participants pointed out that it can be challenging to click on a range block, mostly when used in an inline function block (for example, range $E$16 in Figure 6.3). If the click is not precisely on the range block, the function block is selected instead.

Visibility
All participants were able to see the complete formulas at a glance. To understand the formula, it is also essential to see which cells are referenced, which can be problematic in a large spreadsheet. If users inspect a formula in the formula bar, they have to scroll to the cells they can not see. If a user selects a range block in XLBlocks, it will automatically scroll to the corresponding cell in the spreadsheet.

Several participants commented that multiple nested binary operations (see, for example, the else clause in Figure 6.4b) were difficult to understand. Function blocks have different colors, depending on their category. All binary functions have the same category (Math and Trigonometry) and, therefore, the same color. Furthermore, each function block has its own border, but it is thin and subtly colored. These issues combined make it difficult to distinguish the individual functions.

Premature Commitment
In XLBlocks, the user can build a formula in any order. Also, it is possible to change the order of the functions within the formula at any given time. The only requirement is that the output of the top-level function is connected to a formula block.

Nevertheless, some participants had the feeling they had to start with the formula block and build the formula from there, starting with the top-level function. They even said they liked how the blocks would force them to build the formula in a structured manner. For example, P1 said: *IJou are somewhat forced in a structure, and I actually like that.”*, and P7: *”I think, because it is visual, you are forced to think about the formula you are building.”*

Other participants recognized that they could start anywhere in the formula. All participants agreed that it is easy to change the order of functions in the formula.

Hidden Dependencies

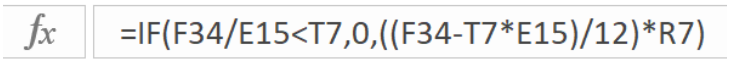
In a spreadsheet, there are dependencies between functions in a formula and between cells in the spreadsheet. Participants indicated that it is easy to see the dependencies between functions in a formula. Each function is visualized as a puzzle piece, and the connection between two pieces visualizes the dependency between the functions.

Concerning the spreadsheet level, participants liked that XLBlocks would highlight all cells in the spreadsheet used in the formula. Also, the possibility to automatically scroll to a precedent cell in the spreadsheet by clicking on the corresponding range block was appreciated. P6 said: *"Clicking on highlighted cells and jumping from one formula to another makes it easier to explain the spreadsheet"*, or according to P7: *IJou can click on a cell and immediately jump to that formula, that for me is one of the biggest advantages."* Furthermore, they liked the option that once a precedent cell was selected in XLBlocks, they could inspect that cell's formula in XLBlocks.

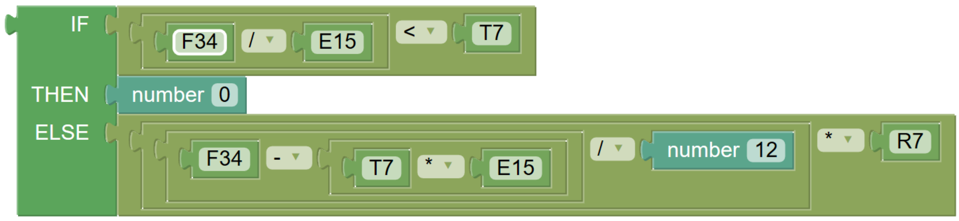
Unfortunately, once they jumped to one of the precedent cells, it was impossible to jump back to the original formula. They also indicated that it was not possible in XLBlocks to see depending cells of a formula. When they had to trace dependents during the comprehension tasks, they had to fall back to Excel's native trace dependents function.

Role-Expressiveness

Participants said that in XLBlocks, they could 'see' the formula. They named the IF function as an example. *"I think you will make fewer logic errors because you really see the formula"* (P6), *"The structure is clear, it is more transparent, you see it immediately."* (P11), and *"It is very easy to read the IF, THEN, ELSE formula."* (P21). In the formula bar (see Figure 6.4a), the IF function is displayed as a single string. A comma separates the THEN and ELSE parts, and by convention, the second argument is the THEN part, and the third argument the ELSE part. In XLBlocks (see Figure 6.4b), the IF, THEN, and ELSE parts are labeled and visualized on three different lines. Furthermore, each block has a thin border that acts as the equivalent of parentheses. According to the participants, these features combined made it easier to understand the formula.


fx =IF(F34/E15<T7,0,((F34-T7*E15)/12)*R7)

(a) The IF function as a long string in the formula bar



(b) The IF function split over several lines with named parameters in XLBlocks

Figuur 6.4: Two variant of the IF Function

XLBlocks, like the formula bar, does not provide an explanation about the function parameters. When working with the VLOOKUP function, participants indicated that they were unsure about the meaning of some of the parameters, and explanation would have helped. This could easily be solved in XLBlocks by adding tool-tips to the parameters.

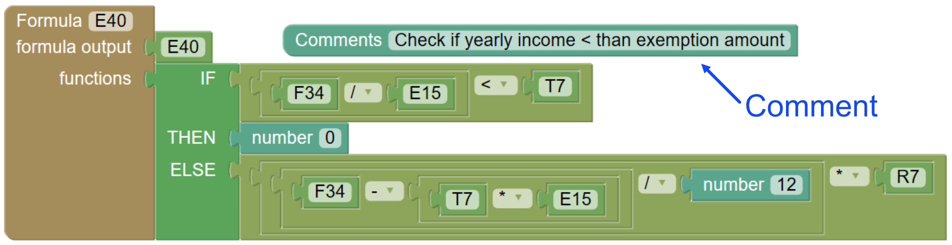
Error-Proneness

As was already mentioned at *Premature Commitment*, some participants pointed out that the blocks guide you through a function's structure. It is not possible to forget a parameter and, because they are labeled, one can not confuse them. As a consequence, this leads to fewer errors. Additionally, most participants noted that XLBlocks places the parentheses and commas automatically, which further reduces possible errors.

Some participants remarked that the operators in the binary function blocks (see Figure 6.7b) are hard to read, and it is easy to forget to change the default operator. Both cases would lead to an incorrect formula.

Secondary Notation

XLBlocks has a dedicated comment block (see Figure 6.5) that can be used to annotate a formula. Multiple comment blocks can be added to a formula, and they can be placed freely on the canvas.

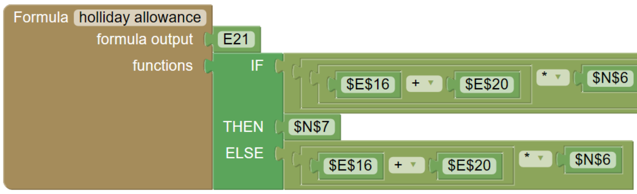


Figuur 6.5: Example of a comment block

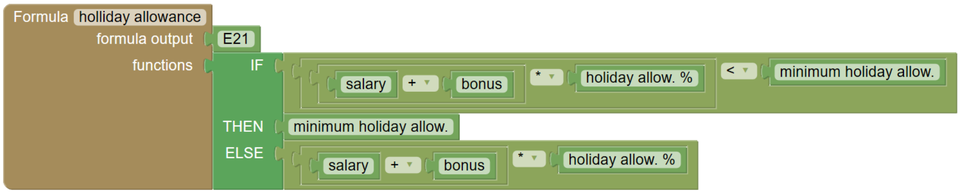
Participants said that they would use the comment block to document the formula, describe its purpose, explain the calculation, and describe the meaning of the cells used in the calculation. The current comment block has been designed to accommodate short comments, but several participants indicated they would like to enter larger text blocks.

Closeness of Mapping

Cells used in a formula are visualized in XLBlocks with range blocks (see Figure 6.3). The cells are identified by their cell address in A1 notation. When participants had to explain formulas, they had to look up the cell by their address in the spreadsheet to see its functional meaning. While explaining a complex formula, it occurred more than once that they forgot the meaning of a cell and had to look it up again, which would slow down the explanation. Several participants mentioned they would prefer the possibility to give the range blocks a meaningful name. An example can be found in Figure 6.6. Figure 6.6a shows the formula as it is currently visualized in XLBlocks, and in Figure 6.6b the cell addresses are replaced by meaningful names.



(a) Cell references in A1 style



(b) Cell references with meaningful names

Figuur 6.6: Different styles for cell references

Consistency

Participants recognized that blocks with the same purpose have the same color. Constants have a different color than cell references, and Lookup functions have a different color than logical functions. The use of color helped them to understand the formulas better. Some participants remarked that it would be even better if the colors would be repeated in the toolbox menu. In that case, it is easier to derive the meaning of a specific color.

There are over 450 functions in Excel. It is not feasible to give every function a unique color. For that reason, functions of the same type (as defined by Microsoft[6], such as math and trigonometry, logical, and lookup and reference) have the same color. If in a formula several functions of the same type are nested (for example, the addition, multiplication, and comparison in the IF clause of Figure 6.6a), this will lead to a group of blocks with the same color. According to the participants, this makes it less easy to read and interpreter the formula. Some participants argued that maybe the color of a function block should depend on the formula's level of nestedness.

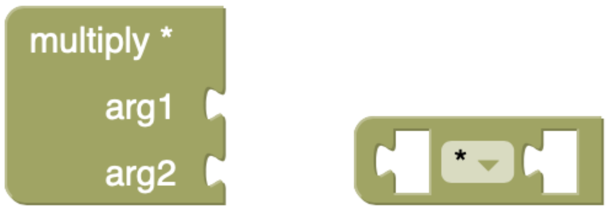
If a user selects (a part) of a formula in XLBlocks, the used cells are highlighted in the spreadsheet. These cells get all the same highlight color. Some participants suggested giving each highlighted cell its own color and using that color to highlight the formula's corresponding range block.

Diffuseness

According to the participants, the size of the formulas is adequate. This is remarkable because block-based languages tend to be more diffuse [85] and programmers value that as a negative because less code will fit on the screen. However, spreadsheet users are accustomed to entering their formulas in a tiny formula bar. They are relieved that in XLBlocks, they have more space available, and because XLBlocks focuses on one formula

[6]https://support.microsoft.com/en-us/office/excel-functions-alphabetical-b3944572-255d-4efb-bb96-c6d90033e188

at a time, even complex formulas will fit on the canvas and do not require additional scrolling.



(a) External (b) Inline

Figuur 6.7: Different input types

XLBlocks handles the input of a function block in two different ways: external (Figure 6.7a) and inline (Figure 6.7b). Basic functions such as addition, subtraction, and division use the inline variant. The inline variant is more natural to read, but if several of these functions are concatenated, one ends up with a wide formula block (see, for example, the else clause in Figure 6.4b). The participants confirmed that several binary operations after each other take more space than needed. According to the participants, also the constant blocks (number, text, and boolean) are relatively large in relation to their importance in the formula (see, for example, the number block in Figure 6.4b).

Hard Mental Operations

According to the participants, working with XLBlocks does not require more mental effort than working with formulas in the formula bar. They provide two reasons for this:

- Because of the visualization in blocks, the formula is split into smaller components. This makes it easier to understand the formula. Participants do not have to understand the formula at once but can focus on a smaller component. For example, P1 noted: *"I do not have to split the formula into smaller parts, XLBlocks does that for me, which makes it easier to understand."*

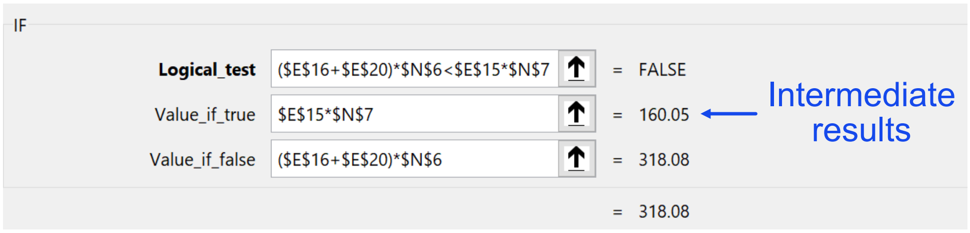
- In XLBlocks, the user does not have to think about the placement of parentheses, quotes, or commas.

Provisionality

Participants loved the fact that they could play with formulas in XLBlocks. P13: *"Dragging a part of formula out of the formula is very easy"* and P19: *"It is very visual, you can drag a part out of it and paste it back in very easily, I really like that."*

They could easily drag components out of the formula onto the canvas and replace them with other functions or components. A component that is dragged out of a formula can be parked and saved on the canvas. It will not influence the generation of the spreadsheet formula. If they were not satisfied with their changes, they could quickly revert to the previous state. It gave them also the opportunity to evaluate two or more variations of the same formula. Finally, they liked the opportunity to change the order of functions within a formula easily.

Progressive Evaluation
Participants indicated that it is possible to stop working on a formula at any time. The formula does not need to be correct before it can be saved, which is not the case in Excel's formula bar. It is possible to test a part of a formula as long as it will lead to a valid formula expression. Seeing if a formula was finished was easy, according to the participants. As long as there are no missing puzzle pieces, the formula is finished. In Excel's formula bar, it is much harder to see if an argument of a function, a parenthesis, or comma is missing.


IF

Logical_test	($E$16+$E$20)*$N$6<$E$15*$N$7	= FALSE
Value_if_true	$E$15*$N$7	= 160.05
Value_if_false	($E$16+$E$20)*$N$6	= 318.08

= 318.08

Figuur 6.8: Formula wizard in Excel displaying intermediate results

The participants indicated that they were missing intermediate results in XLBlocks. This is possible in the formula wizard in Excel (see Figure 6.8), and they would like to see similar functionality in XLBlocks.

6.3.3 Research Questions
In the final paragraphs of this section, we will answer the research questions.

RQ1 understand a formula
Participants believe that, for complex formulas, it is easier to understand them with XLBlocks than with the formula bar. They have several arguments for this. First, the formula's block-based representation splits the formula into smaller components, making it easier to comprehend. Secondly, the parameters of a function are named. Less knowledge of the function syntax is needed to understand it, and lastly, if the user clicks on the formula in XLBlocks, all cells used in the formula are highlighted in the spreadsheet. This enables them to see which numbers are used in the calculation.

RQ2 explain a formula to somebody else
According to the participants, XLBlocks supports the user in explaining the formula. When they click on a part of a formula, XLBlocks highlight the blocks in that part of the formula. This helps in focusing the explanation on a specific part of the formula. Furthermore, comment blocks can be used to document the purpose of a formula and explain components in the formula, such as explaining the test performed in an IF statement. Finally, by clicking on the individual range blocks in the formula, participants could navigate the spreadsheet, highlighting the cells used in the formula and explaining the numbers' functional meaning in these cells.

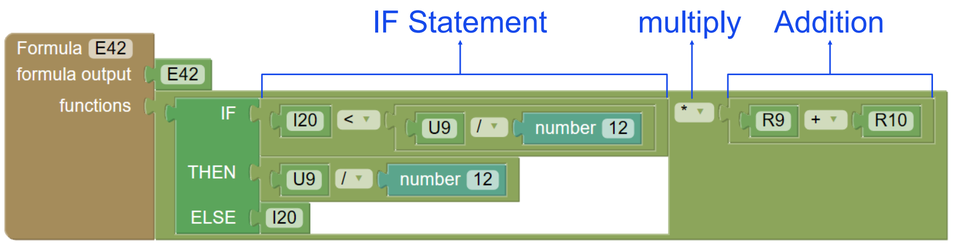
RQ3 understand the spreadsheet model as a whole
XLblocks focuses on a single formula at a time. Nevertheless, participants were able to get an understanding of the working of the spreadsheet as a whole. Even if they were

not familiar with the functional domain of the spreadsheet (payroll administration). Participants could easily click from one formula to another to see how the different formulas were related to each other and form an opinion about the workings of the model. Also, the possibility of automatically scrolling to the different cells and quickly reading the labels helped to understand the functional meaning of the calculation.

6.4 Discussion

6.4.1 Confusing IF statement

During the think-aloud study, several participants got confused when explaining a formula that contained an IF function (see Figure 6.9). In this formula, the outcome of an IF function is multiplied with the addition of two percentages. Because of how the blocks are visualized, the multiplication of the sum of two percentages is displayed at the same height as the IF statement. This led the participants to believe that the multiplication was a part of the IF statement, leading to a logical test that did not make any sense when translated into business terms.



Figuur 6.9: Several participants struggled to explain this formula

Several factors are causing this misconception.

- Both the operand and the arguments of the binary function are aligned at the top. If the multiplication symbol and the addition of the two percentages had been aligned at the middle of the block, the formula would be less confusing.
- Except for the IF function itself, all other functions in the formula are binary functions and have the same color, making it difficult to distinguish them from each other.
- Each function has its own border, but it is very thin with a light gray color to simulate a 3D effect. This makes it challenging to see where one function ends, and another begins.

Aligning the operands and arguments at the middle of a block and making the borders of function a fraction thicker with a more contrasting color would prevent this kind of misconception.

6.4.2 Giving Range Blocks Meaningful Names

As described in the previous section, several participants suggested that the readability of formulas would benefit if range blocks got meaningful names instead of an abstract cell address (see also Figure 6.6b). In Excel, it is possible to assign names to ranges, and several authors have advocated the use of range names [103, 104]. However, other studies indicate that there are inherent risks in using named ranges. Panko and Ordway [1] identify the risk that named ranges can appear correct in the formula but refer to the wrong range, and McKeever [105, 106] found that the use of named ranges decreases the ability of novice spreadsheet users to find and correct errors. The question, if it would be possible to implement range names in XLBlocks in such a way that it would not hinder the debugging process and makes it easy to see to which range the name refers, would make for interesting future work.

6.4.3 Navigating Formulas

While performing task *T06 Determine relationships between two cells*, participants noted that it is easy to navigate in XLBlocks to a direct precedent of the current formula but not to navigate back. This could be solved in XLBlocks by displaying a breadcrumb trail at the top of XLBlocks' canvas. It would show a horizontal list of formulas that the user has analyzed, and by clicking on any of the formulas in the list, the user would navigate back to that formula. Another solution that could be implemented complementary to the breadcrumb trail would be a browser-like navigate back button.

6.4.4 Intermediate Results

When answering questions about the cognitive dimension *Progressive Evaluation* (see Section 6.3.2), participants indicated that it would be even easier to evaluate formulas if, in XLBlocks, they could see the intermediate results of the calculation. Inspired by the work of Leber *et. al* [107], Figure 6.10 shows an example of how this could be implemented in XLBlocks.

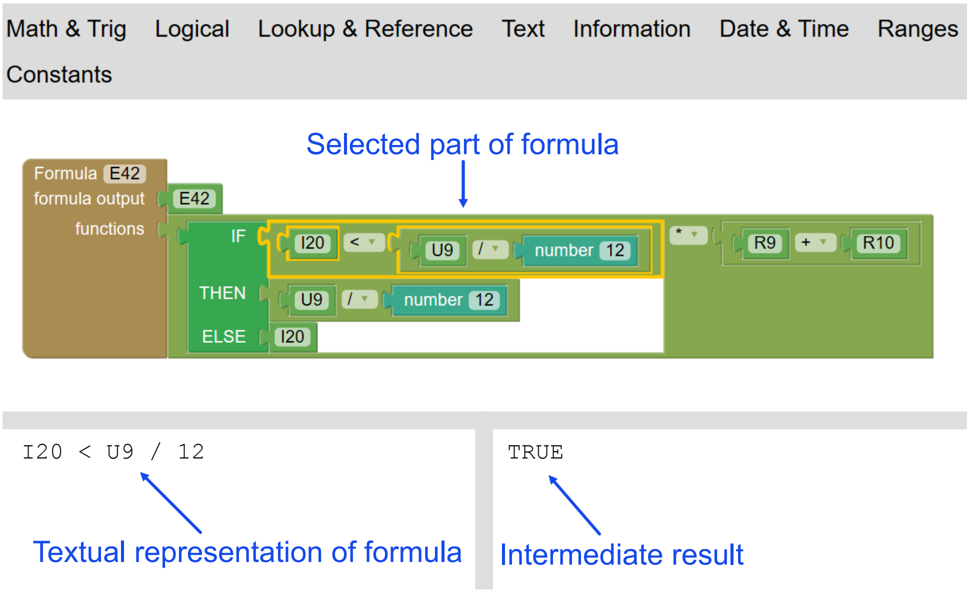
If the user selects a part of a formula, a textual representation of that part of the formula is displayed at the bottom left of the XLBlocks interface, while on the bottom right, the result of the calculation is displayed. One could even consider making the textual representation of the formula also editable. We keep this as a point for future work.

6.4.5 Threats to Validity

A threat to the external validity of our think-aloud study concerns the representativeness of the participants. Additional studies are necessary to generalize our findings.

Furthermore, there is a risk of aptitude treatment interaction since participants of a previous study were also invited to participate in this study. It could be the case that only the most positive ones responded to this request. Eventually, only five of the twenty-one participants were also a participant in the previous study, and judging by the number of points of criticism we received from them during the think-aloud study and the interview, we believe our finding were not impacted by the aptitude treatment.

Another threat to the external validity is the representativeness of the comprehension tasks. We mitigated this by using a validated set of comprehension tasks defined by Pacione *et. al* [52].



Figuur 6.10: Showing textual formula and intermediate results in XLBlocks

Participants were selected from our network, which is a threat to the internal validity of our study. However, we believe the current group serves as a useful reference group, as the persons were experienced professional spreadsheet users, came from different companies, and worked in different functional domains.

We fulfill the role of both developer of XLBlocks and an interviewer during the think-aloud study. This is a threat to the internal validity of the study. We lessened this risk by using a validated set of comprehension tasks and using the CDN Framework to guide our questions during the interview, ensuring that all aspects of the usability of the XLBlocks interface were covered.

6.5 Concluding Remarks

The purpose of this paper is to research the effect of a block-based language on formula comprehension in spreadsheets. We extended the block-based formula editor XLBlocks with functionality to generate block-based representations of existing spreadsheet formulas. We asked participants in a think-aloud study to perform twelve comprehension task and immediately after they finished these tasks, interviewed them about their experience with XLBlocks.

Participants believed that XLBlocks helped them to understand formulas better. They argued that the formula's visualization in blocks helped separate smaller parts in the formula, making it easier to comprehend. Furthermore, in XLBlocks, each function argument had a descriptive label, making the formula more comfortable to read. Finally, the formula's cells were also highlighted in the spreadsheet, making it easier to see what was calculated by the formula.

During the study, participants had to explain three different formulas using XLBlocks. According to participants, XLBlocks made it easier to do so. They could select a part of a formula during an explanation, and XLBlocks would highlight the relevant blocks. This helped both the participants and the listener to focus their attention on this part of the formula. Participants also noted that it is possible to document a formula's workings with the dedicated comment blocks of XLBlocks. Finally, if a formula was part of a larger calculation chain, users could easily navigate the formula's precedents to show how the formulas worked together.

This mechanic of navigating between formulas was, according to the participants, also instrumental in gaining an understanding of the spreadsheet's workings. The possibility of quickly navigating to cells in the spreadsheets by selecting a cell reference in XLBlocks and looking up the corresponding labels in the spreadsheet helped to gain more insight into the spreadsheet's problem domain.

This research gives rise to several directions for future work. Tracing relations between formulas is instrumental in understanding the workings of a spreadsheet. We have seen that XLBlocks can help in finding precedents of a formula but lacks functionality in tracing dependents. We will investigate what would be the best way to extend XLBlocks with this functionality. Furthermore, we will extend XLBlocks with the possibility to show the textual representation of (a part of) the formula and the corresponding intermediate results of the calculation in real-time. It will enable users to receive direct feedback on changes they make in the formula. Finally, we will explore the possibility of giving cell references in XLBlocks a meaningful name.

6.6 Data Availability

A video of XLBlocks, the source code of XLBlocks, the spreadsheet model used during the Think-Aloud study, and the interview questions used are available on figshare, DOI 10.6084/m9.figshare.14268017

7 Conclusion

In this chapter, we present the conclusions of this dissertation. Our research can roughly be divided into two phases. In phase one, we started our research by getting a better understanding of how users interact with spreadsheets and what causes the error-proneness of spreadsheets. We looked at the occurrence of code smells in spreadsheets and if they lead to problematic spreadsheets (Chapter 2). Besides formulas, we also researched how the data and formulas can be organized and how that impacts comprehensibility. Finally, we explored how spreadsheets evolve (Chapter 4).

Based on our research in phase one, we concluded that almost all errors in spreadsheets originated in formulas and decided to focus our research in phase two on finding ways to improve how formulas can be created, edited, and analyzed. This led to the development of a visual language: XLBlocks (Chapter 5). With the first prototype of XLBlocks, it was possible to create formulas with a block-based language. XLBlocks would translate the block-based formula to a textual representation and insert it in the spreadsheet. However, it was not possible to go the other way around. Therefore we extended XLBlocks with the functionality to translate the textual representation of an existing formula to a block-model in XLBlocks (Chapter 6).

In the remainder of this chapter we will describe the contribution of our research (Section 7.1), explore avenues for future research (Section 7.2), and present our conclusions by reflecting on the thesis statement introduced in Chapter 1 (Section 7.3).

7.1 Contributions

The contributions of this dissertation are:

- An approach to conduct a pair-wise comparison of spreadsheets on the occurrence of smells, size metrics, and coupling metrics (Chapter 2)
- Defining a ground truth about the smelliness of a spreadsheet (Chapter 2)
- A definition of the concept of delocalized plans in spreadsheets (Chapter 3)
- Design of a controlled experiment to analyze the ability of spreadsheet users to comprehend and adapt a spreadsheet (Chapter 3)

- Translation of the software comprehension tasks as defined by Pacione *et al.* [52] to the spreadsheet domain (Chapter 3)
- An empirical evaluation of the effect of delocalized plans in spreadsheets on spreadsheet comprehension (Chapter 3)
- FormulaMatch: an algorithm to match unique formulas of two different versions of the same spreadsheet (Chapter 4)
- Two case studies in which a detailed study is made of the evolution of a spreadsheet. In both cases, the analyzed spreadsheets had a time span of three years (Chapter 4)
- Insights from spreadsheet users about how the result of an evolution study can support them in creating better spreadsheets (Chapter 4)
- XLBlocks: a block-based formula editor (Chapter 5 & 6)
- Two think-aloud studies with professional and experienced spreadsheet users to evaluate the use of XLBlocks to create, edit and comprehend spreadsheet formulas (Chapter 5 & 6)
- Two evaluations of the visual language XLBlocks with the Cognitive Dimensions of Notation framework (Chapter 5 & 6)
- A code-generator that translates the textual representation of a spreadsheet formula to an XML-definition of the block-based representation of that formula (Chapter 6)

7.2 Future Work

While conducting the research for this dissertation, we encountered several topics that warrant future work.

7.2.1 Navigating the calculation chain

In this thesis, much attention has been paid to the individual formulas in a spreadsheet. However, for more complex calculations, a single formula is just a step in the calculation. One could compare it with a single line of code in a program. It is the complete calculation chain of a formula that could be compared with a function or routine within a program. In both Chapter 2 and Chapter 3, we saw the importance of the calculation chain on comprehension. There are two specific areas concerning calculation chains that lend themselves to further research.

Firstly, code smells have been defined both on the level of formulas [27] and on the level of worksheets [28]. Would it be possible to define and recognize code smells in calculation chains? Moreover, do these calculation chain smells differ from formula or (inter)worksheet smells?

Secondly, to comprehend what kind of calculations are made in a calculation chain, a user has to analyze the individual formulas of the calculation chain. To do so, a user first needs to recognize which formulas belong to the chain and then navigate in the correct order from one formula to another. We saw the importance of navigating a spreadsheet in

Chapter 3 and Chapter 6. It would be interesting to see if we can assist the user in recognizing the calculation chain and find easy ways to navigate through it. Also, especially if the calculation chain is part of a delocalized plan in a spreadsheet (Chapter 3), it would be beneficial if all elements of the calculation chain could be presented to the user in a single glance. Some work in this area has been done by Roy *et. al.* [108], but the challenge that remains is how to support users in comprehending and navigating complex calculation chains.

7.2.2 Naming in spreadsheets

The default way for naming a cell in a spreadsheet is the A1 notation. A1 refers to the cell in column A and row one. An alternative notation is R1C1, which refers to the cell in row one and column one and is equivalent to A1. R[-1]C[0] is a relative reference. It means to go back one row and stay in the same column as the cell where this reference is used. We have used the R1C1 notation to identify unique formulas. Finally, spreadsheets have the opportunity to give meaningful names to individual cells or a group of cells. These are the so-called named ranges.

In XLBlocks, we refer to other cells with the A1 notation. In both think-aloud studies in Chapter 5 and 6, we received the feedback that users thought it would be easier if they could give meaningful names to cell references. There is some research on the effect of naming, but there is no consensus about which notation is better. Proponents believe that assigning meaningful names to ranges makes it easier to comprehend formulas [103] [104]. On the other hand, Panko and Ordway [1] argue that maybe it is easier to read a formula, but it is less transparent to which cells the names are referring. The formula could look OK, but at the same time, the name could refer to the wrong cells. Furthermore, studies conducted by McKeever *et. al.* indicate that named ranges hinder novice spreadsheet users in their debugging performance [105] [106]. Further research on the naming of cells and ranges is required to understand the impact on comprehension and error-proneness.

7.2.3 Version Management

In Chapter 4 we described the challenges one faces when analyzing changes that were made between two versions of the same spreadsheet. The challenges are twofold. First, a simple structural change like inserting a row or removing a column can lead to changes in sometimes thousands of formulas, while from a functional point of view, nothing was changed. Second, the habit of spreadsheet users to copy formulas down or to the right leads to a high number of formulas that do precisely the same, but because of the A1 notation, are different. If the user decides to change the formula for such a set of copied cells, it will again lead to many changes, while only one calculation was changed. In Chapter 4 we attempted to solve the second problem with FormulaMatch. Using FormulaMatch helps to detect changes in unique formulas. We think it could be further improved if it is augmented with algorithms that detect the structural changes (inserting or deleting rows and columns) that were made.

7.2.4 Further XLBlocks Improvements

During the think-aloud studies about XLBlocks, we received positive reactions from professional and experienced Excel users. The prevailing impression was that XLBlocks would be a valuable tool to create, edit, and analyze complex formulas. Some participants asked if they could already get a copy of the research prototype.

However, the feedback gave us also an extensive list of possible improvements. Most proposed improvements are engineering questions, but some of the improvements give rise to further research.

Naming

As described in one of the earlier paragraphs in this Chapter, users indicated they would prefer to refer to cells with meaningful names instead of the abstract A1 notation. It would lead to more readable formulas. However, although naming cells sounds attractive, one of the disadvantages of this approach is that it removes the relation between the reference (A1) and the location (row one, column A). If users read a formula with names instead of cell addresses, they may have a better understanding of what kind of calculation is made but may have no idea where the cells that are used in that calculation are located. As stated in Section 7.2.2, research is needed to understand if using names lead to better comprehension and fewer errors, and if it is possible to work with names without losing the relation to the location of the cells.

Tracing the calculation chain

In the current implementation of XLBlocks, it is possible to navigate from a formula to its precedents. The opposite, navigating to dependents of a formula, is not possible and should be added. Adding this functionality should ideally take into account the results of the research we proposed in Section 7.2.1. Not only should it be easy to jump back and forth from formula to formula, but somehow the position of the formula in the complete calculation chain should be visualized.

Making XLBlocks Hybrid

XLBlocks is a visual, block-based language. The professional Excel users who participated in our think-aloud studies have a long history with a formula's textual representation. They liked the block-based representation of formulas, but while working with XLBlocks, they were missing the textual representation. They would prefer to construct the formula using XLBlocks, but at the same time see the textual representation.

At the moment, it is only possible to see the textual representation when the creation of a formula in XLBlocks is finished. However, according to the participants, it would add value if they could see the textual formula during construction in XLBlocks. It would be even better if they could also choose to edit the textual representation. As they said, some changes are easier to make in the block model, while other changes could be easier accomplished by editing text.

Another possible improvement that is closely related to this hybrid form of XLBlocks is not only simultaneously displaying the blocks-based and textual representations but also adding the intermediate results of the different steps in the calculation.

Quantitative evaluation of XLBlocks

Our research on XLBlocks in the last two chapters was qualitative and exploratory. This was a well-considered choice. We were in the early phase of the development of XLBlocks and wanted to understand how professional users experience the use of XLBlocks. A more quantitative approach can help further development of XLBlocks. Will the use of XLBlocks lead to fewer errors? Will editing of complex formulas be faster? Will it help users to understand spreadsheets better? These questions could be answered with similar controlled experiments as we used in Chapter 3.

7.3 Reflections on the thesis statement

In Chapter 1 we described the thesis statement that guided our research:

> *A **visual language** supports **professional spreadsheet users** in **interacting** with **complex formulas**. This results in a **reduction in the number of errors** made during the **creation** or **maintenance** of formulas.*

In this concluding chapter, we reflect on this statement. We will confront several elements of this statement with the findings of our research.

7.3.1 Professional Spreadsheet Users

An overarching goal of our research is to support professional spreadsheet users. As described in Chapter 1, the impact of spreadsheet errors is most felt in an industrial setting. For that reason, we tried to involve professional spreadsheet users in our research and use, whenever possible, real-life spreadsheets in our studies.

Our analysis of smells (Chapter 2) was based on a set of 54 spreadsheets created by customers of the financial modeling company F1F9 and 54 spreadsheets created by the professional modelers of F1F9.

In our experiment on the effect of delocalized plans in spreadsheets (Chapters 3), we recruited the participants from participants of one of our MOOCs about spreadsheets and the mailing lists of EUSPRIG and our research website. All sources make it likely that prospective participants are interested in spreadsheets and have more than average spreadsheet skills. This turned out to be true. We asked the participants to rate themselves on their perceived skill level of Excel, and they scored on average 3.6 on a five-point Likert scale.

We conducted our spreadsheet evaluation study (Chapter 4) together with Alliander, an industrial partner. We used two different sets of spreadsheets from Alliander that were used for internal reporting about occurrences of failures in the gas distribution network and the medium voltage electricity grid of the Netherlands. Both spreadsheet models were in use for a period of three years.

In the think-aloud studies that we used in our evaluations of XLBlocks (Chapter 5 & 6), we worked with a relatively small number of participants. We recruited them from within our network of industrial partners, and we were looking for participants that use spreadsheets in their daily work, have a more than average skill level, and have multi-year experience. Twenty-eight spreadsheet users with an average of twenty-five years of experience participated. We asked them to rate their own skill level on a scale from one to ten, and on average, they scored themselves an eight.

7.3.2 Complex Formulas

We started our research by getting a better understanding of the term complexity in spreadsheets. There are two approaches to analyze complexity in spreadsheets: 1) Use metrics to measure the complexity [40] or 2) analyze the occurrence of code smells [27] [28] [38]. As shown in Chapter 2, we found that size and coupling metrics do not succeed in differentiating between problematic and normal spreadsheets. They are not good metrics to measure spreadsheet complexity.

In [27], [28], and [38] we find successful translations of code smells to the domain of spreadsheets. However, these studies do not demonstrate that the presence of smells, by definition, leads to problematic spreadsheets. In Chapter 2 we compared 54 pairs of spreadsheets, each consisting of a spreadsheet that was perceived as problematic by its owners and a similar version of that spreadsheet that was remodeled according to best practices by professional financial model builders. We found that the remodeled spreadsheets suffered from smells to a much lower extent than the problematic spreadsheets and concluded that the occurrence of smells is indeed an indicator for problematic spreadsheets.

For code smells in spreadsheets, there is a trade-off between the smells Multiple Operations and Multiple References (both indications of long formulas) on the one hand, and Long Calculation Chain on the other hand [27]. In Chapter 2 we observed this trade-off in action. In the F1F9 (less smelly) spreadsheets, the occurrence of the Multiple Operations and Multiple Reference smells decreased, while at the same time, the occurrence of the Long Calculation Chain increased. With a constant business complexity, smaller formulas mean that calculations must be split into smaller steps, leading to longer calculation chains. This begs the question of what is more important, making the formula smaller or reducing the length of the calculation chain. In Chapter 3 we found an answer. Reducing the length of the calculation chain has a more (positive) impact on the maintainability of a formula than the length of the formula itself.

7.3.3 Interacting

In Chapter 3 and 4, we shifted our focus from formulas to the way spreadsheet users interact with the spreadsheet.

In Chapter 3 we explore how data in spreadsheets are organized. We translated the concept of locality in source code to spreadsheets. Often a calculation in a spreadsheet does not consist of a single formula. In these cases, the calculation is divided into several steps. A delocalized plan in a spreadsheet is a multi-step calculation with steps that are spread across the spreadsheet (instead of located closely together). We found that spreadsheet users perform significantly better on comprehension tasks when spreadsheets contain less delocalized plans.

In Chapter 4 we focus our research on how users interact with spreadsheets that they have to maintain for multiple years. We study the evolution of two sets of spreadsheets that span three years. From the two case studies, we learned that spreadsheets grow over time, both in data and the number of formulas. Also, we found that in almost every version of the spreadsheets, changes were made to some of the formulas. The main reasons for these changes were: 1) new feature requests, 2) improved maintainability, and 3) fixing bugs.

To analyze the differences between the individual spreadsheets within the evolution sets, we developed FormulaMatch. This algorithm makes it possible to find matching unique formulas in two spreadsheets and determine if they have been changed (or if no match is found, either created or deleted). Spreadsheet users involved in the case studies indicated that it would help them if an automatic evolution analysis could provide them with:

- a summary of all changes (so that they can check if they made all necessary adjustments)
- an overview of frequently changed formulas (acting as a checklist for changes that have to be made)
- suggestions for refactoring (e.g., listing formulas that have to be adjusted every time data is added to the model)
- highlight drops or spikes in changing and growing rates (which are often an indication something went wrong during the last update)

7.3.4 Visual Language

In Chapter 2 through 4 we focused our research on spreadsheet complexity. The findings of these studies, together with research on spreadsheet errors [73] [7] brought us to a conclusion that, although spreadsheets consist of data, layout, and formulas, most spreadsheet errors have their origin in formulas.

That is why we shifted our research focus to the way formulas are created and edited in spreadsheets. To make correct formulas, spreadsheet users need to know the exact syntax of a function. Errors caused by forgetting or misplacing a comma, parenthesis, or quotes are easily made. These problems with the formula syntax are similar to the problems novice programmers encounter when they start programming. Research has shown that their performance can be improved if they can use a visual instead of a textual language [74] [75].

We hypothesize that a visual language for creating and editing spreadsheet formulas could support spreadsheet users in a similar way. For that reason, we developed XLBlocks. A block-based formula editor for spreadsheets. With XLBlocks, users can create new formulas (Chapter 5) or inspect and modify existing formulas (Chapter 6).

7.3.5 Creation

In Chapter 5 we introduce XLBlocks. In this study, we evaluate the use of XLBlocks as a formula editor for spreadsheets. We focus in this study on the creation of formulas.

We noted that users appreciated the space available on the canvas to create formulas. For us, this was surprising because, in comparisons between block-based and textual languages, one of the disadvantages of block-based languages mentioned is the space it occupies. However, spreadsheet users are used to the tiny space they have available in the default formula bar, and they welcome the space available in XLBlocks.

A second advantage that the participants mentioned is that they could start anywhere in the formula. If one creates a nested formula in the formula bar in Excel, one must carefully think about with which function one should start. Once entering the formula (in the formula bar), it is challenging to change the order of the functions, and if a mistake is made and the syntax of the formula is incorrect, it is not possible to save the formula.

This is not the case in XLBlocks. Because of the drag and drop interface, it is effortless to change the order of the functions in the formula. XLBlocks will take care of the correct placement of parentheses, quotes, and commas. According to the users, the function blocks guided them through the function's syntax. All arguments were visible at a glance, they were always in the proper order, and a missing puzzle piece would indicate a forgotten argument.

During the think-aloud study, we observed that because participants could freely drag and drop parts of the formula on the canvas, it helped them think about it. They started with the elements they were sure about and continued with less clear functions. They kept playing with the different elements until it became clear how the complete formula should be constructed.

Participants also mentioned that they liked the possibility of dragging out a part of a formula and replacing it with a different part to try out different variants of a formula. They appreciated that there was no need to clean up the canvas.

Finally, they mentioned that, unlike in Excel and other spreadsheet tools, it was possible to save an unfinished (and even incorrect) formula in XLBlocks.

7.3.6 Maintenance

From prior research, we know that spreadsheets have an average lifespan of five years, and during that period, they are used by thirteen different users [6]. A more extended lifespan makes maintenance likely, and because of the different users, knowledge about the spreadsheet needs to transfer from one user to another. For both maintenance and knowledge transfer, spreadsheet comprehension is critical. We, therefore, focus in Chapter 6 on the effect of XLBlocks on formula comprehension. For that reason, we extended XLBlocks with the functionality to generate a block-based representation of existing formulas.

In the think-aloud study that we conducted, we asked participants to make changes to an existing spreadsheet model and explain the working of several formulas.

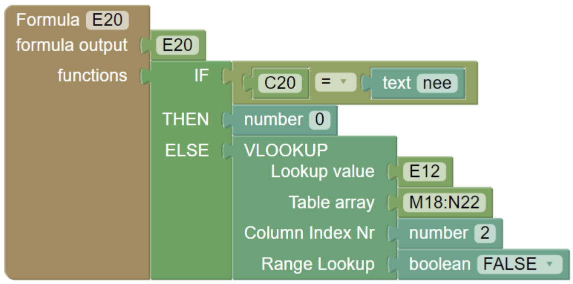
We found that users find it easy to make changes to existing formulas with XLBlocks. Because functions are visualized in blocks, it is easy to locate the formula part that needs to be adjusted. Furthermore, they do not have to think about parentheses, quotes, and commas. XLBlocks handles that for them.

In order to gain an understanding of formula comprehension, we asked participants during the think-aloud study to explain the working of several formulas. We observed that several properties of XLBlocks made it easier for the participants to understand the formula.

First of all, in contrast to the formula bar, in XLBlocks, the arguments of a function are named (see Figure 7.1). This means that the participants did not need to know the order of the arguments of a function.

Furthermore, if a user selects a function in XLBlocks, all cells referenced in that function are simultaneously highlighted in the spreadsheet. It makes it easy for the user to see which cells are used by the formula, which makes it easier to determine what kind of calculation is made by the formula.

Several participants mentioned that because every function in a formula is a separate block, XLBlocks splits a formula automatically into smaller parts. This makes it easier



Figuur 7.1: Example of named function arguments in IF and VLOOKUP functions

to comprehend the formula. It also literally visualizes the nested structure of a complex formula.

We not only asked the participants to explain the working of formulas, but we also asked them to explain the working of the spreadsheet as a whole. Being able to understand the meaning of the individual formulas helps to get an understanding of the spreadsheet. However, this was not the only reason why XLBlocks supported the user in understanding the spreadsheet. In XLBlocks, the user can click on a cell reference block, and in the spreadsheet, the cursor automatically jumps to that cell. It allows the user to read the label of that cell, and we observed that this mechanic helped the participants quickly jump from formula to formula, get a sense of the context in which a formula was used and form an idea about how the spreadsheet as a whole was working.

7.3.7 Reduction in the Number of Errors

We end this section by revisiting the question of whether a visual language reduces the number of errors made in formulas. In the think-aloud studies about XLBlocks, our primary goal was to learn from the participants' experiences with the language. Based on the participants' feedback, we think several factors reduce the chance of making errors.

Most obvious is the automatic placement of commas, parentheses, and quotes. Also, because functions are visualized as little puzzle pieces, it is difficult to forget an argument of a function. If this happens, it is implicitly visualized by a missing puzzle piece. Because the function's arguments have to be connected to the function itself, the user cannot specify the arguments in the wrong order. These are properties of XLBlocks that directly affect the error-proneness of formulas.

However, other properties also affect error proneness:

- the possibility to start anywhere in a formula
- the drag and drop interface
- the feature to easily change the order of functions in a formula
- the opportunity to try out different variants of the formula

All the above help reduce the mental effort needed to create or edit a formula, which helps reduce the chance that errors are made.

Based on the feedback received and the observations made during the think-aloud studies, we are convinced that:

A visual language supports professional spreadsheet users in interacting with complex formulas. This results in a reduction in the number of errors made during the creation or maintenance of formulas.

Bibliography

Bibliografie

[1] Raymond R Panko and Nicholas Ordway. Sarbanes-oxley: What about all the spreadsheets? *arXiv preprint arXiv:0804.0797*, 2008.

[2] WL Winston. Executive education opportunities millions of analysts need training in spreadsheet modeling, optimization, monte carlo simulation and data analysis. *OR MS TODAY*, 28(4):36–39, 2001.

[3] USA Bureau of Labor Statistics. Computer and internet use at work in 2003. `https://www.bls.gov/news.release/pdf/ciuaw.pdf`, August 2005. Accessed: 2021-10-08.

[4] Kevin Taylor. An analysis of computer use across 95 organisations in europe, north america and australasia. `https://wellnomics.com/wp-content/uploads/2020/01/Wellnomics-white-paper-Comparison-of-Computer-Use-across-different-Countries.pdf`, August 2007. Accessed: 2021-10-08.

[5] Jonathan P Caulkins, Erica Layne Morrison, and Timothy Weidemann. Do spreadsheet errors lead to bad decisions? perspectives of executives and senior managers. In *Evolutionary Concepts in End User Productivity and Performance: Applications for Organizational Progress*, pages 44–62. IGI Global, 2009.

[6] Felienne Hermans, Martin Pinzger, and Arie van Deursen. Supporting professional spreadsheet users by generating leveled dataflow diagrams. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 451–460. ACM, 2011.

[7] Raymond R Panko. What we know about spreadsheet errors. *Journal of Organizational and End User Computing (JOEUC)*, 10(2):15–21, 1998.

[8] Diane Janvrin and Joline Morrison. Using a structured design approach to reduce risks in end user spreadsheet development. *Information & management*, 37(1):1–12, 2000.

[9] Raymond R Panko and Richard P Halverson Jr. Are two heads better than one?(at reducing errors in spreadsheet modeling). *Office Systems Research Journal*, 15(1):21–32, 1997.

[10] Felienne Hermans and Emerson Murphy-Hill. Enron's spreadsheets and related emails: A dataset and analysis. In *Proceedings of the 37th International Conference on Software Engineering-Volume 2*, pages 7–16. IEEE Press, 2015.

[11] Raymond R Panko. Spreadsheet errors: What we know. what we think we can do. In *Proceedings of the Spreadsheet Risk Symposium*, 2000.

[12] Carmen M Reinhart and Kenneth S Rogoff. Growth in a time of debt. *American economic review*, 100(2):573–78, 2010.

[13] Thomas Herndon, Michael Ash, and Robert Pollin. Does high public debt consistently stifle economic growth? a critique of reinhart and rogoff. *Cambridge journal of economics*, 38(2):257–279, 2014.

[14] Felienne Hermans, Bas Jansen, Sohon Roy, Efthimia Aivaloglou, Alaaeddin Swidan, and David Hoepelman. Spreadsheets are code: An overview of software engineering approaches applied to spreadsheets. In *Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering*, 2016.

[15] Felienne Hermans and Danny Dig. Bumblebee: a refactoring environment for spreadsheet formulas. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 747–750. ACM, 2014.

[16] Sandro Badame and Danny Dig. Refactoring meets spreadsheet formulas. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 399–409. IEEE, 2012.

[17] Jorma Sajaniemi. Modeling spreadsheet audit: A rigorous approach to automatic visualization. *Journal of Visual Languages & Computing*, 11(1):49–82, 2000.

[18] AAS Swidan. *Challenges of end-user programmers: Reflections from two groups of end-users*. PhD thesis, Delft University of Technology, 2019.

[19] Martin Erwig, Robin Abraham, Irene Cooperstein, and Steve Kollmansberger. Automatic generation and maintenance of correct spreadsheets. In *Proceedings of the 27th international conference on Software engineering*, pages 136–145. ACM, 2005.

[20] Sohon Roy, Felienne Hermans, and Arie van Deursen. Spreadsheet testing in practice. In *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*, pages 338–348. IEEE, 2017.

[21] Bas Jansen and Felienne Hermans. Using a visual language to create better spreadsheets. *Software Engineering Methods in Spreadsheets*, 2014.

[22] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why don't software developers use static analysis tools to find bugs? In *2013 35th International Conference on Software Engineering (ICSE)*, pages 672–681. IEEE, 2013.

[23] Norman Fenton and James Bieman. *Software Metrics: A Rigorous and Practical Approach, Third Edition*. CRC Press, Inc., USA, 3rd edition, 2014.

[24] Robert K Yin. *Case study research: Design and methods*, volume 5. sage, 2009.

[25] K Anders Ericsson and Herbert A Simon. *Protocol analysis: Verbal reports as data*. the MIT Press, 1984.

[26] Alan F Blackwell, Carol Britton, A Cox, Thomas RG Green, Corin Gurr, Gada Kadoda, MS Kutar, Martin Loomes, Chrystopher L Nehaniv, Marian Petre, et al. Cognitive dimensions of notations: Design tools for cognitive technology. In *International Conference on Cognitive Technology*, pages 325–341. Springer, 2001.

[27] Felienne Hermans, Martin Pinzger, and Arie van Deursen. Detecting code smells in spreadsheet formulas. *Proceedings of the International Conference on Software Maintenance (ICSM)*, 2012.

[28] Felienne Hermans, Martin Pinzger, and Arie van Deursen. Detecting and visualizing inter-worksheet smells in spreadsheets. In *Proceedings of the 2012 International Conference on Software Engineering*, pages 441–451. IEEE Press, 2012.

[29] Bas Jansen and Felienne Hermans. Code smells in spreadsheet formulas revisited on an industrial dataset. In *Proceedings of the 2015 International Conference on Software Maintenance and Evolution*, pages 372–380. IEEE Press, 2015.

[30] Bas Jansen and Felienne Hermans. The effect of delocalized plans on spreadsheet comprehension: a controlled experiment. In *Proceedings of the 25th International Conference on Program Comprehension*, pages 286–296. IEEE Press, 2017.

[31] B. Jansen, F. Hermans, and E. Tazelaar. Detecting and predicting evolution in spreadsheets - a case study in an energy network company. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 645–654, Sep. 2018.

[32] Bas Jansen and Felienne Hermans. Xlblocks: a block-based formula editor for spreadsheet formulas. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 55–63. IEEE, 2019.

[33] David Weintrop, Heather Killen, Talal Munzar, and Baker Franke. Block-based comprehension: Exploring and explaining student outcomes from a read-only block-based exam. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, pages 1218–1224, New York, NY, USA, 2019. Association for Computing Machinery.

[34] Bas Jansen and Felienne Hermans. The effect of a block-based language on formula comprehension in spreadsheets. In *Proceedings of the 29th international conference on Software Engineeringth International Conference on Program Comprehension*. IEEE Press, 2021.

[35] Martin Fowler. *Refactoring : improving the design of existing code.* Addison-Wesley, Reading, MA, 1999.

[36] FAST Standard Organisation. The fast standard - practical, structured design rules for financial modelling, version fast02a, 2015.

[37] Felienne Hermans. *Analyzing and visualizing Spreadsheets.* PhD thesis, PhD thesis, Software Engineering Research Group, Delft University of Technology, Netherlands, 2012.

[38] Jácome Cunha, João P Fernandes, Hugo Ribeiro, and João Saraiva. Towards a catalog of spreadsheet smells. In *Computational Science and Its Applications–ICCSA 2012*, pages 202–216. Springer, 2012.

[39] Bas Jansen. Enron versus euses: A comparison of two spreadsheet corpora. In *Proceedings of the 2nd Workshop on Software Engineering Methods in Spreadsheets, Florence, Italy*, 2015.

[40] A. Bregar. Complexity metrics for spreadsheet models. In *Proc. of EuSpRIG '04*, page 9, 2004.

[41] K. Hodnigg and R.T. Mittermeir. Metrics-based spreadsheet visualization: Support for focused maintenance. In *Proc. of EuSpRIG '08*, page 16, 2008.

[42] Stephen Hole, Duncan McPhee, and Alex Lohfink. Mining spreadsheet complexity data to classify end user developers. In *Proceedings of the 2009 International Conference on Data Mining (DMIN)*, pages 573–579, 2009.

[43] Felienne Hermans, Ben Sedee, Martin Pinzger, and Arie van Deursen. Data clone detection and visualization in spreadsheets. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 292–301. IEEE Press, 2013.

[44] Karen J Rothermel, Curtis R Cook, Margaret M Burnett, Justin Schonfeld, Thomas RG Green, and Gregg Rothermel. Wysiwyt testing in the spreadsheet paradigm: An empirical evaluation. In *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, pages 230–239. IEEE, 2000.

[45] Felienne Hermans, Martin Pinzger, and Arie van Deursen. *ECOOP 2010 – Object-Oriented Programming: 24th European Conference, Maribor, Slovenia, June 21-25, 2010. Proceedings*, chapter Automatically Extracting Class Diagrams from Spreadsheets, pages 52–75. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[46] Jácome Cunha, Martin Erwig, and Joao Saraiva. Automatically inferring classsheet models from spreadsheets. In *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on*, pages 93–100. IEEE, 2010.

[47] Gerald M Weinberg. *The psychology of computer programming*, volume 932633420. Van Nostrand Reinhold New York, 1971.

[48] Stanley Letovsky and Elliot Soloway. Delocalized plans and program comprehension. *IEEE Software*, 3(3):41, 1986.

[49] Daqing Hou, Chandan Raj Rupakheti, and H James Hoover. Documenting and evaluating scattered concerns for framework usability: A case study. In *2008 15th Asia-Pacific Software Engineering Conference*, pages 213–220. IEEE, 2008.

[50] Larry L Constantine. Visual coherence and usability: a cohesion metric for assessing the quality of dialogue and screen designs. In *Computer-Human Interaction, 1996. Proceedings., Sixth Australian Conference on*, pages 115–121. IEEE, 1996.

[51] DG Conway and CT Ragsdale. Modeling optimization problems in the unstructured world of spreadsheets. *Omega*, 25(3):313–322, 1997.

[52] Michael J Pacione, Marc Roper, and Murray Wood. A novel software visualisation model to support software comprehension. In *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*, pages 70–79. IEEE, 2004.

[53] Felienne Hermans and Efthimia Aivaloglou. Do code smells hamper novice programming? a controlled experiment on scratch programs. In *Program Comprehension (ICPC), 2016 IEEE 24th International Conference on*, pages 1–10. IEEE, 2016.

[54] Bas Cornelissen, Andy Zaidman, and Arie Van Deursen. A controlled experiment for program comprehension through trace visualization. *Software Engineering, IEEE Transactions on*, 37(3):341–355, 2011.

[55] Tiago L Alves, Christiaan Ypma, and Joost Visser. Deriving metric thresholds from benchmark data. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10. IEEE, 2010.

[56] Jeanine Romano, Jeffrey D Kromrey, Jesse Coraggio, and Jeff Skowronek. Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the NSSE and other surveys. In *annual meeting of the Florida Association of Institutional Research*, pages 1–33, 2006.

[57] Felienne Hermans, Martin Pinzger, and Arie van Deursen. Detecting and refactoring code smells in spreadsheet formulas. *Empirical Software Engineering*, pages 1–27, 2014.

[58] Daniel W Barowy, Dimitar Gochev, and Emery D Berger. Checkcell: Data debugging for spreadsheets. In *ACM SIGPLAN Notices*, volume 49, pages 507–523. ACM, 2014.

[59] Meir M Lehman and Juan F Ramil. Software evolution—background, theory, practice. *Information Processing Letters*, 88(1-2):33–44, 2003.

[60] Renato Lima Novais, André Torres, Thiago Souto Mendes, Manoel Mendonça, and Nico Zazworka. Software evolution visualization: A systematic mapping study. *Information and Software Technology*, 55(11):1860–1883, 2013.

[61] Harald Gall, Mehdi Jazayeri, Rene R Klosch, and Georg Trausmuth. Software evolution observations based on product release history. In *Software Maintenance, 1997. Proceedings., International Conference on*, pages 160–166. IEEE, 1997.

[62] Wensheng Dou, Liang Xu, Shing-Chi Cheung, Chushu Gao, Jun Wei, and Tao Huang. Venron: a versioned spreadsheet corpus and related evolution analysis. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 162–171. ACM, 2016.

[63] Meir M Lehman. The programming process. *internal IBM report*, 1969.

[64] Meir M Lehman. Programs, cities, students—limits to growth? In *Programming Methodology*, pages 42–69. Springer, 1978.

[65] Bryan Klimt and Yiming Yang. The Enron corpus: A new dataset for email classification research. In *European conference on machine learning*, pages 217–226. Springer, 2004.

[66] Liang Xu, Wensheng Dou, Chushu Gao, Jie Wang, Jun Wei, Hua Zhong, and Tao Huang. Spreadcluster: recovering versioned spreadsheets through similarity-based clustering. In *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on*, pages 158–169. IEEE, 2017.

[67] Chris Chambers, Martin Erwig, and Markus Luckey. Sheetdiff: A tool for identifying changes in spreadsheets. In *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on*, pages 85–92. IEEE, 2010.

[68] Anna Harutyunyan, Glencora Borradaile, Christopher Chambers, and Christopher Scaffidi. Planted-model evaluation of algorithms for identifying differences between spreadsheets. In *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on*, pages 7–14. IEEE, 2012.

[69] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[70] Beat Fluri, Michael Wuersch, Martin PInzger, and Harald Gall. Change distilling: Tree differencing for fine-grained source code change extraction. *IEEE Transactions on software engineering*, 33(11), 2007.

[71] David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[72] John W Creswell and J David Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches.* Sage publications, 2017.

[73] R. R. Panko and R. P. Halverson. Spreadsheets on trial: a survey of research on spreadsheet risks. In *Proceedings of HICSS-29: 29th Hawaii International Conference on System Sciences*, volume 2, pages 326–335 vol.2, Jan 1996.

[74] Thomas W Price and Tiffany Barnes. Comparing textual and block interfaces in a novice programming environment. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*, pages 91–99. ACM, 2015.

[75] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. Scratch: Programming for all. *Commun. ACM*, 52(11):60–67, November 2009.

[76] Microsoft. Javascript API for Office. `https://docs.microsoft.com/en-us/office/dev/add-ins/reference/javascript-api-for-office`. Accessed: 2021-01-29.

[77] N. Fraser. Ten things we've learned from blockly. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, pages 49–50, Oct 2015.

[78] F. Hermans, E. Aivaloglou, and B. Jansen. Detecting problematic lookup functions in spreadsheets. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 153–157, Oct 2015.

[79] Efthimia Aivaloglou, David Hoepelman, and Felienne Hermans. Parsing excel formulas: A grammar and its application on 4 large datasets. *Journal of Software: Evolution and Process*, 29(12):e1895, 2017.

[80] Matt Bellingham, Simon Holland, and Paul Mulholland. A cognitive dimensions analysis of interaction design for algorithmic composition software. In *Proceedings of Psychology of Programming Interest Group Annual Conference 2014*, pages 135–140. University of Sussex, 2014.

[81] Marty Kauhanen and Robert Biddle. Cognitive dimensions of a game scripting tool. In *Proceedings of the 2007 conference on Future Play*, pages 97–104. ACM, 2007.

[82] Franklyn Turbak, David Wolber, and Paul Medlock-Walton. The design of naming features in app inventor 2. In *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 129–132. IEEE, 2014.

[83] Robin Abraham, Martin Erwig, Steve Kollmansberger, and Ethan Seifert. Visual specifications of correct spreadsheets. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 189–196. IEEE, 2005.

[84] Alan F Blackwell and Thomas RG Green. A cognitive dimensions questionnaire optimised for users. In *Proceedings of Psychology of Programming Interest Group (PPIG) Annual Conference 2000*, volume 13, pages 137–154, 2000.

[85] Thomas R. G. Green and Marian Petre. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages & Computing*, 7(2):131–174, 1996.

[86] Thomas Green and Alan Blackwell. Cognitive dimensions of information artefacts: a tutorial. In *BCS HCI Conference*, volume 98, pages 1–75, 1998.

[87] David Weintrop and Uri Wilensky. To block or not to block, that is the question: Students' perceptions of blocks-based programming. In *Proceedings of the 14th International Conference on Interaction Design and Children*, IDC '15, pages 199–208, New York, NY, USA, 2015. ACM.

[88] David Bau, Jeff Gray, Caitlin Kelleher, Josh Sheldon, and Franklyn A. Turbak. Learnable programming: Blocks and beyond. *CoRR*, abs/1705.09413, 2017.

[89] Google. Blockly Toolbox Guide. `https://developers.google.com/blockly/guides/configure/web/toolbox#shadow_blocks`. Accessed: 2019-05-06.

[90] MIT. Scratch Backpack. `https://en.scratch-wiki.info/wiki/Backpack`. Accessed: 2019-05-06.

[91] Margaret M. Burnett, J. William Atwood, Rebecca Walpole Djang, James Reichwein, Herkimer J. Gottfried, and Sherry Yang. Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. *Journal of functional programming*, 11(2):155–206, 2001.

[92] Roxanne Leitão and Chris Roast. Developing visualisations for spreadsheet formulae: towards increasing the accessibility of science, technology, engineering and maths subjects. In *Joint Proceedings of the MathUI, OpenMath and ThEdu Workshops and Work in Progress track at CICM co-located with Conferences on Intelligent Computer Mathematics (CICM 2014)*, 2014.

[93] A. Sarkar, A. D. Gordon, S. P. Jones, and N. Toronto. Calculation view: multiple-representation editing in spreadsheets. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 85–93, Oct 2018.

[94] Gregor Engels and Martin Erwig. Classsheets: automatic generation of spreadsheet applications from object-oriented specifications. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 124–133. ACM, 2005.

[95] J. Cunha, J. Mendes, J. Saraiva, and J. P. Fernandes. Embedding and evolution of spreadsheet models in spreadsheet systems. In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 179–186, Sep. 2011.

[96] Ephraim P Glinert. Towards second generation interactive graphical programming environments. In *Proceedings of IEEE Workshop onVisual Language. IEEE CS Press, Silver Spring, MD*, pages 61–70, 1986.

[97] Matthew Conway, Randy Pausch, Rich Gossweiler, and Tommy Burnette. Alice: a rapid prototyping system for building virtual environments. In *CHI Conference Companion*, pages 295–296. ACM, 1994.

[98] David Weintrop, Afsoon Afzal, Jean Salac, Patrick Francis, Boyang Li, David C. Shepherd, and Diana Franklin. Evaluating coblox: A comparative study of robotics programming environments for adult novices. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 366:1–366:12, New York, NY, USA, 2018. ACM.

[99] R. Holwerda and F. Hermans. A usability analysis of blocks-based programming editors using cognitive dimensions. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 217–225, Oct 2018.

[100] R. Holwerda and F. Hermans. Towards blocks-based prototyping of web applications. In *2017 IEEE Blocks and Beyond Workshop (B B)*, pages 41–44, Oct 2017.

[101] XLParser web demo. `https://xlparser.perfectxl.nl/demo/`. Accessed: 2021-01-29.

[102] Blockly. `https://developers.google.com/blockly`. Accessed: 2021-01-29.

[103] SE Kruck. Testing spreadsheet accuracy theory. *Information and Software Technology*, 48(3):204–213, 2006.

[104] Philip L Bewig. How do you know your spreadsheet is right? *arXiv preprint arXiv:1301.5878*, 2013.

[105] Ruth McKeever, Kevin McDaid, and Brian Bishop. An exploratory analysis of the impact of named ranges on the debugging performance of novice users. In *Proceedings of the 2009 European Spreadsheet Risk Interest Group (EuSpRiG) Conference*, 2009.

[106] Ruth McKeever and Kevin McDaid. How do range names hinder novice spreadsheet debugging performance? In *Proceedings of the 2010 European Spreadsheet Risk Interest Group (EuSpRiG) Conference*, 2010.

[107] Ž. Leber, M. Črepinek, and T. Kosar. Simultaneous multiple representation editing environment for primary school education. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 175–179, 2019.

[108] S. Roy, F.F.J. Hermans, and A. van Deursen. Supporting spreadsheet maintenance with dependency notification. In *Proceedings of 6th International Symposium on End-User Development (IS-EUD 2017)*, pages 88–93. Technische Universiteit Eindhoven, 2017.

Curriculum Vitæ

Bas Jansen

1971/12/14	Date of birth in Arnhem, The Netherlands

Education

2013-2022	Ph.D. Student, Software Engineering Research Group, Delft University of Technology, The Netherlands, *XLBlocks: On the Effect of a Visual Language on Formula Creation and Comprehension in Spreadsheets* Supervisor: Dr. ir. Felienne Hermans Promotor: Prof. Dr. Arie van Deursen
2003-2007	Executive Master of Finance and Control, Post Graduate program for Controllers, VU University Amsterdam
1990-1996	Master of Science Industrial Engineering & Management, University of Twente
1984-1990	University Entrance Diploma, Christelijk Lyceum Arnhem

Experience

2008-today	Owner & Information Consultant, InfoAction, Zaltbommel
2004-2008	Business Controller, CIAD, Culemborg
2000-2004	Manager & Senior Consultant, KPMG Optimum, Zaltbommel
1996-2000	Project Manager & Consultant, Akzo Nobel Information Systems, Arnhem

Academic Service

Reviewer	Computer Languages, Systems & Structures, 2016
	Information and Software Technology, 2016
	ICSME, 2016
	ICSME, 2017
	ICSME, 2018
	Information and Software Technology, 2018
	The Journal of Systems and Software, 2021

List of Publications

🗎 11. *Bas Jansen*, Felienne Hermans: The Effect of a Block-based Language on Formula Comprehension in Spreadsheets. In the Proceedings of the 29th International Conference on Program Comprehension (ICPC). 2021.

🗎 10. *Bas Jansen*, Felienne Hermans: XLBlocks: A block-based formula editor for spreadsheet formulas. In the Proceedings of the 2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Memphis (USA), 2019.

🗎 09. *Bas Jansen*, Felienne Hermans, Edwin Tazelaar: Detecting and Predicting Evolution in Spreadsheets: A Case Study in an Energy Network Company. In the Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), Madrid (Spain), 2018

08. *Bas Jansen*, Felienne Hermans: The Use of Charts, Pivot Tables and Array Formulas in two Popular Spreadsheet Corpora. In the Proceedings of the 5th International Workshop on Software Engineering Methods in Spreadsheets (SEMS), Lisbon (Portugal), 2018.

🗎 07. *Bas Jansen*, Felienne Hermans: The Effect of Delocalized Plans on Spreadsheet Comprehension: A Controlled Experiment. In the Proceedings of 25th International Conference on Program Comprehension (ICPC), Buenos Aires, Argentina, 2017.

06. *Bas Jansen:* Polaris: Providing Context Aware Navigation in Spreadsheets. In the Prodceedings of the 2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Cambridge (UK), 2016.

05. Felienne Hermans, *Bas Jansen*, Sohon Roy, Efthimia Aivaloglou, Alaaeddin Swidan, David Hoepelman: Spreadsheets are Code: An Overview of Software Engineering Approaches applied to Spreadsheets. In the Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Osaka (Japan), 2016.

04. Felienne Hermans, Efthimia Aivaloglou, *Bas Jansen*: Detecting Problematic Lookup Functions in Spreadsheets. In the Proceedings of 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Atlanta (USA), 2015

🗎 03. *Bas Jansen*, Felienne Hermans: Code Smells in Spreadsheet Formulas Revisited on an Industrial Dataset. In the Proceedings of the 31st International Conference on Software Maintenance and Evolution (ICSME), Bremen (Germany), 2015.

02. *Bas Jansen*, Felienne Hermans: Enron versus EUSES: A Comparison of two Spreadsheet Corpora. In the Proceedings of the 2nd Workshop on Software Engineering Methods in Spreadsheets (SEMS), Florence (Italy), 2015.

01. *Bas Jansen*, Felienne Hermans: Using a visual language to create better spreadsheets. In the Proceedings of the 1st International Workshop on Software Engineering Methods in Spreadsheets (SEMS), Delft (The Netherlands), 2014.

🗎 Included in this thesis.